GFA Software

GFA Systemtechnik GmbH
Heerdter Sandberg 30-32
D-4000 Düsseldorf 11
Telefon 0211/5504-0

GFA
SYSTEMTECHNIK

AMIGA

GFA BASIC 3.0

Interpreter

Alle Rechte vorbehalten. Kein Teil dieses Handbuches darf ohne schriftliche Genehmigung der GFA Systemtechnik GmbH, in welcher Form und mit welchen Mitteln auch immer, sei es auf elektronischem oder mechanischem Weg, kopiert, vervielfältigt oder übermittelt werden. Dies gilt auch für Fotokopien und Tonbandaufnahmen, es sei denn, sie seien für den persönlichen Gebrauch des Käufers bestimmt.

Das auf der beiliegenden Diskette enthaltene Programm GFA-BASIC 3.0 Interpreter darf, außer zum persönlichen Gebrauch des Käufers, nicht kopiert werden. Der Run-Only-Interpreter GFABASRO dagegen darf beliebig weitergegeben werden.

Der Verlag hat alle Sorgfalt walten lassen, um vollständige und akkurate Informationen zu veröffentlichen. GFA Systemtechnik übernimmt keine Gewähr dafür, daß die beschriebenen Verfahren, Programme usw. funktionsfähig und frei von Schutzrechten Dritter sind.

Dieses Handbuch wurde mit Ventura Publisher 1.1 erstellt und auf Linotronic 300 belichtet. Satz und Druck erfolgte durch Graf & Pflügge, Düsseldorf.

© Copyright 1988 GFA Systemtechnik GmbH

Programmautor: Frank Ostrowski

Änderungen des Textes vorbehalten, Inhalt ohne Gewähr.

2. Auflage, Dezember 1988

Einleitung und Editor

Variablen und Speicherverwaltung

Operatoren

Numerische Funktionen und Zeichenketten

Ein- und Ausgabe

Disketten und Dateiverwaltung

Programmsteuerung

Grafik

Ereignis-, Menü-, Screen- und Fensterverwaltung

Fortgeschrittene Programmierung

Systemroutinen

Anhang

Erweiterte Version GFA-BASIC 3. 5 Amiga

Zum Handbuch **GFA-BASIC 3.0 Interpreter** sind folgende Kapitel hinzuzufügen:

- **Kapitel 13 - Lineare Operationen mit Vektoren und Matrizen (Seite 13 - 1 bis 13 - 41)**

- **Kapitel 14 - Weitere Befehle der Version 3.5 (Seite 14 - 1 bis 14 - 8)**

- **Kapitel 1 - Einleitung, Der Editor:**
  Geänderte Editoreigenschaften im GFA-BASIC 3.5 (Seite 1 - 25)

- Im **Inhaltsverzeichnis** sind die **erste** und **letzte** Seite auszutauschen.

Inhaltsverzeichnis

1 - Einleitung und Editor

Vorwort 1-2
Arbeiten mit dem Handbuch 1-3
Erste Schritte mit GFA-BASIC 3.0 1-6
Der Editor 1-8
Der Cursor-Tasten-Block 1-9
Der Zifferntastenblock 1-11
Control-Kommandos 1-12
Weitere Editiertasten 1-13
Die Menüleiste und die Funktionstasten 1-14
Die Zeilenanzeige und die Uhr 1-22
Geänderte Editoreigenschaften im BASIC 3.5 1-25

2 - Variablen und Speicherverwaltung

Variablentypen 2-2
Felder 2-4
DEFxxx 2-5
LET 2-6
CLEAR / CLR 2-7
ERASE 2-8
DIM 2-9
OPTION BASE 2-10
INSERT / DELETE 2-11
ARRAYFILL 2-12
ASC / CHR$ 2-13
BIN$ / OCT$ / HEX$ 2-14
STR$ 2-15
VAL 2-16
CINT / CFLOAT 2-17
CVx / MKx 2-18
SWAP 2-20
QSORT / SSORT 2-22
FALSE / TRUE / PI 2-25
DATE$ / TIME$ 2-26
SETTIME 2-27
TIMER 2-28
FRE 2-29
TYPE 2-30
DUMP 2-31

VARPTR / ARRPTR 2-33
* 2-34
POKE 2-35
PEEK 2-36
{x} 2-37
ABSOLUTE 2-39
VOID 2-40
MALLOC / MFREE 2-41
BMOVE 2-43
RESERVE 2-44

3 - Operatoren

Operatorenhierarchie 3-2
Logische Operatoren 3-3
NOT 3-4
AND 3-5
OR 3-6
XOR 3-7
IMP 3-8
EQV 3-9
+/- 3-10
=/= = ... 3-11
< >/< = >/= 3-14
< > 3-16

4 - Numerische Funktionen und Zeichenketten

ADD/SUB/MUL/DIV/MOD 4-2
SIN/COS/TAN 4-5
SINQ/COSQ 4-6
DEG/RAD 4-7
SQR 4-8
EXP/LOG 4-9
DEC/INC 4-10
PRED/SUCC 4-11
ABS/SGN 4-12
INT/TRUNC/FIX/FRAC 4-13
ROUND 4-14
ODD/EVEN 4-15
MIN/MAX 4-16
RND/RANDOM 4-17
BCLR/BSET/BCHG/BTST 4-19

SHL / SHR 4–21
ROL / ROR 4–22
AND / OR / XOR 4–23
IMP / EQV 4–24
SWAP 4–25
BYTE / CARD / WORD 4–26
LEFT$ / RIGHT$ 4–27
MID$ 4–28
INSTR 4–30
RINSTR 4–31
STRING$ / SPACE$ / SPC 4–32
PRED / SUCC 4–33
LEN / TRIM 4–34
UPPER$ 4–35
LSET / RSET 4–36

5 – Ein- und Ausgabe

INKEY$ 5–2
INPUT 5–3
LINE INPUT 5–5
FORM INPUT 5–6
PRINT 5–7
PRINT USING 5–9
MODE 5–11
DEFNUM 5–12
CRSxxx / POS / TAB 5–13
DATA 5–15
MOUSE 5–17
LPRINT / LPOS 5–19
HARDCOPY 5–20
SOUND / WAVE 5–22

6 – Disketten- und Dateiverwaltung

Dateispezifikation 6–2
Wahlfreier Zugriff 6–3
CHDIR / DIR$ / DFREE 6–4
DIR / FILES 6–6
EXIST 6–7
OPEN 6–8
LOF / LOC / EOF / CLOSE / TOUCH 6–9
NAME / RENAME / KILL 6–11

BLOAD / BSAVE / BGET / BPUT 6–12
INP / OUT 6–13
INPUT$() 6–14
INPUT # / LINE INPUT 6–15
PRINT # / WRITE # 6–16
STORE / RECALL 6–17
SEEK / RELSEEK 6–19
FIELD 6–20
GET / PUT / RECORD 6–21

7 – Programmsteuerung

Schleifen 7–2
Prozeduren und Funktionen 7–3
IF ... ELSE ... ENDIF 7–4
ELSE IF 7–6
ON GOSUB 7–8
SELECT CASE 7–9
FOR ... TO 7–13
REPEAT UNTIL 7–15
WHILE ... WEND 7–16
DO LOOP 7–17
DO WHILE / DO UNTIL 7–18
EXIT IF 7–20
GOSUB / PROCEDURE 7–21
LOCAL 7–23
@funk / FUNCTION 7–24
DEFFN 7–26
ON BREAK 7–28
ON ERROR 7–29
ERROR / ERR 7–31
EVERY / AFTER 7–32
REM 7–35
GOTO 7–36
PAUSE / DELAY 7–37
END / EDIT / STOP 7–38
NEW 7–39
LOAD 7–40
SAVE / PSAVE 7–41
LIST / LLIST 7–42
CHAIN 7–43
RUN 7–44

SYSTEM / QUIT 7–45
TRON / TROFF 7–46
TRON / TRACE 7–48
$ 7–49
INLINE 7–50
C: 7–51
MONITOR 7–53
CALL 7–54
RCALL 7–55

8 – Grafik

SETCOLOR 8–2
DEFFILL 8–3
BOUNDARY 8–5
DEFLINE 8–6
GRAPHMODE 8–7
RASTPORT 8–8
CLIP 8–9
PLOT / LINE / DRAW 8–10
DRAW / SETDRAW 8–12
BOX / PBOX 8–15
CIRCLE / ELLIPSE 8–16
POLYLINE 8–17
POINT 8–18
FILL 8–19
CLS 8–20
TEXT 8–21
GET / PUT 8–22
VSYNC 8–23
DISPLAY 8–24

9 – Ereignis-, Menü-, Screen- und Fensterverwaltung

Ereignisverwaltung 9–2
Pull-Down-Menüs 9–3
MENU 9–4
MENU() 9–5
ON MENU 9–9
ON MENU... GOSUB 9–10
MENU KILL / MENU X,Y 9–12
Screen-Befehle 9–14
OPENS 9–17

CLOSES 9–19
FRONTS / BACKS 9–20
MOVES 9–21
TITLES 9–22
SETSPEN 9–23
SCREEN 9–24
Fensterbefehle 9–25
OPENW 9–26
CLEARW / TITLEW / BACKW / FULLW 9–32
MOVEW / SIZEW / LIMITW 9–33
SETWPEN 9–34
WINDOW 9–35
ALERT 9–36
FILESELECT 9–37

10 – Fortgeschrittene Programmierung

CLI-Aufruf / _dosCmd$ 10–2
EXEC 10–3
Library-Variablen 10–4
SPRITE 10–5
STICK / STRIG 10–7
SAY / TRANSLATE$() 10–9
FD-Files 10–10

11 – Systemroutinen

Aufruf der Systemroutinen 11–2
Exec-Library 11–3
Graphics-Library 11–24
Workbench-Library 11–54
Intuition-Library 11–58
DiskFont-Library 11–76
Layers-Library 11–78
DOS-Library 11–85

12 – Anhang

Tastatur-Scancodes 12–2
ASCII-Tabelle 12–3
Spezielle ASCII-Zeichen 12–4
Füllmuster 12–6
Amiga-DOS-Fehlermeldungen 12–7
Befehlsverzeichnis 12–8

13 - Lineare Operationen mit Vektoren und Matrizen

Systembefehle 13-1
Erzeugungsbefehle 13-3
Schreib- und Lesebefehle 13-5
Kopier- und Vertauschungsbefehle 13-9
Spezialfälle 13-11
Operationsbefehle 13-25

14 - Weitere Befehle der Version 3.5

Befehle aus dem Bereich der Kombinatorik 14-1
Neue Variante des Befehls OPENW 14-3
Bestimmen von Vorder- und Hintergrundfarbe 14-4
Befehle zur Operation mit DATA-Zeigern 14-5
Spezielle Variablen 14-6

1 – Einleitung und Editor

Vorwort

Mit dem GFA-BASIC-Interpreter 3.0 steht Ihnen eine umfangreiche Programmiersprache mit einer komfortablen Entwicklungsumgebung zur Verfügung.

In modernen Programmiersprachen sind die Möglichkeiten der Programmstrukturierung von besonderer Bedeutung. Bereits der Editor unterstützt die strukturierte Programmierung durch automatische Einrückung von Befehlen in Schleifen und Bedingungen. Darüber hinaus können Unterprogramme im Programm-Listing durch einen Platzhalter vertreten sein, an dessen Stelle durch einen Tastendruck das Unterprogramm „aufgeklappt" werden kann. Im Bereich der bedingten Anweisungen sind zu den Befehlen IF-ELSE-ENDIF, zusätzlich Befehle zur Mehrfachverzweigung nutzbar (ELSE-IF, SELECT-CASE).

Um Unterprogramme formulieren zu können, gibt es die Möglichkeit, Prozeduren und Funktionen zu vereinbaren. Dabei können sowohl Variablenwerte als auch die Variablen selbst übergeben werden. Ferner stehen neben den üblichen Schleifentypen FOR-NEXT, REPEAT-UNTIL, WHILE-WEND und DO-LOOP erweiterte Schleifenkommandos wie DO-UNTIL, DO-WHILE, LOOP-UNTIL und LOOP-WHILE zur Verfügung. Systemnahe Programmierung wird durch die Möglichkeit des Aufrufs von Betriebssystemroutinen gestattet. Viele dieser Funktionen stehen auch in Form einfacher Befehle zur Verfügung. Assembler- und C-Unterprogramme lassen sich mit Befehlen wie z.B. RCALL, C: und Monitor einbinden.

Amiga-Screens und -Windows können durch besonders einfach gehaltene Befehle göffnet und verwaltet werden. Der GFA-BASIC-Interpreter 3.0 verfügt über echte Integer-Arithmetik, die eine hohe Rechengeschwindigkeit bietet, sowie über Fließkomma-Arithmetik mit hoher Rechengenauigkeit (13 Nachkommastellen). Befehle zur Bit-Manipulation (BCLR, BSET, BTST, BCHG, SHL, SHR, ROL, ROR, usw.) fehlen ebensowenig wie eine umfangreiche Sammlung an Variablentypen.

Zuerst folgt eine kurze Erläuterung des Handbuchaufbaus und eine Einführung in die Benutzung des GFA-BASIC-Interpreters.

Darauf werden die Kommandos zur Bedienung des Editors beschrieben. Alle danach folgenden Kapitel beinhalten im wesentlichen eine Beschreibung der Befehle und Funktionen von GFA-BASIC 3.0. Diese sind nach inhaltlichen Gesichtspunkten geordnet. Im Anhang findet sich eine alphabetische Übersicht der Befehle und Funktionen, mit Verweisen auf die entsprechenden Handbuchseiten.

Die Erläuterungen sind wie folgt aufgebaut:

- Syntaxangabe.
- Beschreibung der zulässigen Parametertypen.
- Erläuternder Text.
- Beispiel.

Bei der Syntaxangabe sind optionale Parameter mit eckigen Klammern markiert worden, z.B. so:

```
LEFT$(a$ [,x] )
```

In GFA-BASIC gibt es Befehle und Funktionen. Befehle geben keinen Wert zurück:

```
LINE 100,100,200,200
```

Funktionen liefern einen Rückgabewert. Diesen kann man mit PRINT ausgeben, einer Variablen zuweisen oder in einem Ausdruck verrechnen. Nachfolgend einige Beispiele:

```
PRINT ASC("65")
PRINT ASC("A")
a = ASC("A")
b = ASC("A") + 32
```

In diesem Handbuch wird bei der Syntaxangabe von Funktionen nicht verdeutlicht, daß sie einen Rückgabewert haben, sondern dies geht aus der Beschreibung der Funktionen hervor. Als Syntaxangabe wird dann z.B. nur ASC(a$) angegeben. Wahlfreie Parameter, von denen bei einem Befehl beliebig viele angegeben werden können (z.B. bei DATA), werden ebenfalls mit eckigen Klammern aufgeführt. In diesen stehen zwei Parameter, gefolgt von drei Punkten, z.B.:

DATA [x,y,...]

Die Angabe der erlaubten Parametertypen steht unter der Syntaxangabe. Für diese Typen werden folgende Abkürzungen verwendet:

avar Arithmetische Variable. Dabei muß es sich um eine numerische Variable beliebigen Typs handeln.

aexp Arithmetischer Ausdruck (arithmetic expression). Dies ist ein beliebig komplexer Ausdruck, der eine Zahl produziert. Dabei kann es sich auch um eine Konstante (eine Zahl) oder um eine Variable (Variablen sind eine Teilgruppe der Ausdrücke) handeln. Beispiele für arithmetische Ausdrücke sind:
a%
3
2+a%+ASC("A")

svar String-Variable. Dies ist eine Zeichenkettenvariable. Dieser Variablentyp hat die Endung $.

sexp Zeichenkettenausdruck (string expression). Dieser Ausdruck kann von beliebiger Komplexität sein und muß einen String ergeben. Dabei kann es sich auch um eine Konstante (ein Text in Anführungszeichen) oder um eine Zeichenkettenvariable handeln. Beispiele für String-Ausdrücke sind:
a$
"Test"
a$+"Probe"+LEFT$("HANDBUCH",4)

ivar Ganzzahlige Variable (integer variable).

iexp Ganzzahliger Ausdruck (integer expression).

bexp Logischer Ausdruck (boolean expression).

Wichtig ist, daß bei einigen numerischen Parametern nicht beliebige numerische Variablentypen angegeben werden können. Das wichtigste Beispiel hierfür sind Adressen. Adressen müssen mindestens in einer Vier-Byte-Variable angegeben werden; Boolean-, Byte- oder Word-Variablen sind also nicht zulässig. Nach der Beschreibung der zulässigen Parametertypen folgt die Erläuterung der Befehle. Dort wird die Bedeutung des Befehls und seiner einzelnen Parameter erklärt.

Die Besprechung eines Befehls wird mit einem oder mehreren Beispielen beendet. Jedes dieser Beispiele ist im Editor einzugeben und mit RUN (Shift+F10 oder Anklicken von RUN in der Menüleiste) zu starten. Hinter jedem Beispiel ist angegeben, welchen Effekt es produziert.

Dieses Konzept der Befehlsbeschreibung wurde nur im Abschnitt über die Betriebssystem-Funktionen etwas abgewandelt. Hier wird der Name der entsprechenden Funktion, gefolgt von einer kurzen Beschreibung, angegeben.

Das Handbuch schließt mit einer Sammlung von Tabellen und einer alphabetisch sortierten Liste aller Befehle, mit Verweis auf die jeweilige Seitennummer der Beschreibung.

Erste Schritte mit GFA-BASIC 3.0

Dieser Abschnitt ist für diejenigen Anwender von GFA-BASIC 3.0 gedacht, die bisher noch keine Erfahrungen mit dieser Programmiersprache gemacht haben. Wer also schon mit den ATARI-Versionen gearbeitet hat, kann diesen Abschnitt überschlagen.

Die GFA-BASIC-Programmdiskette ist nicht kopiergeschützt. Erstellen Sie also als erstes eine Sicherheitskopie der Originaldiskette. Die Beschreibung des Kopiervorgangs finden Sie im Handbuch Ihres Rechners. Legen Sie dann die Kopie der Programmdiskette in Ihr Diskettenlaufwerk und starten Sie den GFA-BASIC-Interpreter.

Auf dem Bildschirm erscheint nun der Editor, in dem Sie Ihre Programme schreiben können. Geben Sie nun die folgenden Programmzeilen ein, und drücken Sie nach der Eingabe jeder Zeile die Return-Taste. Dabei brauchen Sie die Leerzeichen, die vor einigen Befehlen stehen, nicht mit einzugeben. Die Einrückungen von Befehlen in Schleifen werden vom Editor automatisch vorgenommen. Es ist auch nicht nötig, bei der Eingabe der Befehlszeilen auf Groß- und Kleinschreibung zu achten. Wenn Sie eine Zeile mit Return verlassen, wird die Groß- und Kleinschreibung automatisch durchgeführt.

```
OPENW 1
REPEAT
  WHILE MOUSEK=1
    BOX MOUSEX,MOUSEY,MOUSEX+30,MOUSEY+30
  WEND
UNTIL MOUSEK=2
```

In der rechten oberen Bildschirmecke finden Sie das Wort „Run". Zeigen Sie mit dem Mauspfeil auf dieses Wort und drücken Sie die linke Maustaste, um das Programm zu starten. Auf dem Workbench-Screen wird ein Fenster geöffnet. Wenn Sie nun die linke Maustaste drücken und die Maus auf dem Tisch bewegen, können Sie in dem Fenster zeichnen.

Als „Pinsel" wird ein Rechteck verwendet. Das Drücken der rechten Maustaste beendet das Programm. Es erscheint dann ein Alert-Requester auf dem Bildschirm, der die Meldung „Programmende" enthält. Zeigen Sie auf das Wort „Return" in diesem Requester und drücken Sie die linke Maustaste. Anschließend befinden Sie sich wieder im Editor.

Wie funktioniert dieses Programm? Das Zeichnen erfolgt mit dem BOX-Befehl in der Mitte des Programms. Er zeichnet Rechtecke. Seine vier Parameter geben die Eckpunkte dieser Rechtecke an. Die Variablen MOUSEK, MOUSEX und MOUSEY beinhalten Informationen über die Maus. In MOUSEK steht, welche Maustaste gerade gedrückt ist. Dabei bedeutet MOUSEK=1, daß die linke und MOUSEK=2, daß die rechte Maustaste gedrückt wird. In MOUSEX und MOUSEY steht die x- bzw. y-Position des Aktionspunktes des Maus-Cursors innerhalb des Fensters.

Die restlichen Befehle (REPEAT, WHILE, WEND, UNTIL) sind Schleifenbefehle. Die Schleife aus den Befehlen WHILE MOUSEK=1 und WEND läßt sich in die Formulierung „wiederhole, solange die linke Maustaste gedrückt ist" übersetzen. Die äußere Schleife aus den Befehlen REPEAT und UNTIL MOUSEK=2 bedeutet soviel wie: „wiederhole, bis die rechte Maustaste gedrückt wird". Da hinter der Zeile mit dem UNTIL-Befehl keine weitere Anweisung mehr folgt, wird das Programm nach dem Verlassen der REPEAT-UNTIL-Schleife beendet.

Gehen Sie nun hinter die letzte Zeile des Programms und geben Sie folgende, absichtlich falsche Programmzeile ein, in der das „i" des Befehlswortes PRINT fehlt:

```
prnt "Test"
```

Wenn Sie nun die Return-Taste drücken, um den Befehl zu bestätigen und die Zeile zu verlassen, so flackert kurz der Bildschirm, und in der zweiten Menüzeile erscheint die Meldung „Syntax Error".

Der Editor prüft also schon während der Programmerstellung, ob die Befehle syntaktisch korrekt eingegeben werden. Gehen Sie nun mit dem Cursor auf den Buchstaben „r" und drücken Sie dreimal die Delete-Taste. In der Programmzeile steht jetzt nur noch:

```
p "Test"
```

Wenn Sie nun die Return-Taste drücken, so können Sie die Zeile verlassen, und der Buchstabe „p" wird automatisch als Abkürzung für das Befehlswort PRINT erkannt.

Der Editor

Der Editor des GFA-BASIC 3.0 ist kein gewöhnlicher Text-Editor, sondern speziell für die Entwicklung von Programmen konzipiert. Dies zeigt sich z.B. darin, daß syntaktisch falsche Befehle bereits während der Programmerstellung erkannt werden. Außerdem werden Befehle in Schleifen oder bedingten Anweisungen automatisch eingerückt und Befehlsabkürzungen in den vollständigen Befehlsnamen überführt (z.B. „p“ in PRINT).

Beim Schreiben eines Programm-Listings wird immer dann, wenn eine Zeile verlassen werden soll, eine Syntaxkontrolle durchgeführt. Ist der Befehl in dieser Zeile syntaktisch nicht korrekt, so erscheint in der zweiten Menüzeile die Meldung „Syntax Error“. Durch das Setzen des Remark-Zeichens (') am Zeilenanfang wird diese Zeile zum Kommentar. Danach ist es möglich, die Zeile zu verlassen. In jede Programmzeile kann nur ein Befehl aufgenommen werden. Hinter diesem Befehl kann lediglich noch ein Remark-Text stehen. Dieser eine Befehl darf nicht mehr als 255 Zeichen lang sein. Wird eine Zeile länger als 80 Zeichen, so verschwindet ihr Anfang am linken Bildschirmrand. Es wird also nur diese eine Zeile horizontal „gescrollt“.

Beim Verlassen einer Befehlszeile wird eine Syntaxkontrolle (s.o.) durchgeführt, die korrekte Einrückung des Befehls vorgenommen und die Zeile formatiert. Dazu gehört z.B. das Entfernen überflüssiger Leerzeichen (z.B. wird aus 2 + 2 dann 2+2) und Verbessern der Schreibweise von Befehlen (groß) und Systemfunktionen.

Die Cursor-Steuerung kann über den Cursor-Tasten-Block erfolgen. Die Tastenbelegung ist:

Pfeil links	→	Cursor ein Zeichen nach links.
Pfeil rechts	→	Cursor ein Zeichen nach rechts.
Pfeil oben	→	Cursor eine Zeile nach oben.
Pfeil unten	→	Cursor eine Zeile nach unten.

Die Bewegungen des Cursors unterliegen dabei einigen Einschränkungen. Er kann maximal ein Zeichen hinter das letzte Zeichen in einer Zeile gesetzt werden und maximal eine Zeile unter die letzte Programmzeile. Wenn der Cursor eine Zeile betritt, die kürzer ist als die aktuelle Cursor-Spaltenposition, so springt der Cursor an das Ende dieser Zeile, ansonsten behält er seine aktuelle Spaltenposition bei. Es ist auch möglich, den Cursor mit der Maus zu positionieren. Zu diesem Zweck zeigt man mit dem Mauspfeil auf die gewünschte Position und drückt die linke Maustaste.

Das Betätigen der Shift+Insert-Taste (Ziffernblock) fügt zwischen der Zeile über dem Cursor und der Cursor-Zeile eine Leerzeile ein, wenn in der aktuellen Zeile keine Änderung vorgenommen wurde. Der Cursor wird an den Anfang dieser Leerzeile gesetzt. Shift+Home bewegt den Cursor in die linke obere Ecke unterhalb der Menüleiste, Control+Home springt an den Anfang des Programm-Listings.

Mit der Help-Taste kann man Änderungen in einer Programmzeile wieder rückgängig machen, sofern die Zeile noch nicht verlassen wurde. Die Help-Taste ermöglicht auch, Prozeduren im Programm-Listing „auf- und zuzuklappen“.

Damit ist folgendes gemeint: Wenn man den Cursor auf eine Zeile positioniert, in der das Befehlswort PROCEDURE steht und dann die Help-Taste drückt, so werden alle Befehlszeilen bis zur nächsten RETURN-Anweisung (einschließlich) nicht mehr im Listing angezeigt. Statt dessen wird vor das Befehlswort PROCEDURE ein eckiger Pfeil und ein Leerzeichen gesetzt „>“.

In dieser Situation lassen sich auch keine Änderungen am Prozedurnamen oder der Parameterliste des Unterprogramms vornehmen. Um die Befehlszeilen zwischen PROCEDURE und RETURN wieder sichtbar zu machen, bewegt man den Cursor auf die mit dem eckigen Pfeil beginnende Zeile und drückt erneut die Help-Taste oder löscht das „>" Zeichen. Dieses „Zusammenklappen" (Folding) eines Unterprogramms ermöglicht es, kurze übersichtliche Listings zu erstellen, in denen man nur das Unterprogramm „aufklappt", an dem man gerade arbeitet. Ein Programm mit zusammengeklappten Prozeduren kann z.B. so aussehen:

```
init
main_menu
> PROCEDURE init
> PROCEDURE main_menu
> PROCEDURE menueleiste
> PROCEDURE laden
> PROCEDURE speichern
> PROCEDURE bearbeiten
> PROCEDURE info_holen
> PROCEDURE info_zeigen
```

Mit einer aufgeklappten Prozedur sieht das Programm dann so aus:

```
init
main_menu
> PROCEDURE init
> PROCEDURE main_menu
> PROCEDURE menueleiste
PROCEDURE laden
  FILESELECT "","Datei laden","OK",datei$
  IF NOT EXIST(datei$)
    ALERT 0,"Datei existiert nicht !",1,"Abbruch",r%
  ELSE
    OPEN "I",#1,datei$
    RECALL #1,da$(),-1,x%
    CLOSE #1
  ENDIF
RETURN
> PROCEDURE speichern
> PROCEDURE bearbeiten
> PROCEDURE info_holen
> PROCEDURE info_zeigen
```

Der Zifferntastenblock dient normalerweise der Eingabe von Ziffern und einiger anderer Zeichen. Er kann aber auch in Verbindung mit der Control-Taste angesprochen werden. Seine Belegung entspricht dann weitgehend der Belegung bei Tastaturen mit eingeschaltetem NUMLOCK (z.B. bei PCs). Die Belegung ist:

Control und 4	Springe an den Zeilenanfang.
Control und 6	Springe an das Zeilenende.
Control und 8	Eine Seite nach oben blättern.
Control und 2	Eine Seite nach unten blättern.
Control und 7	Springe an den Programmanfang.
Control und 1	Springe an das Programmende.
Control und 9	Eine Seite nach oben „blättern“.
Control und 3	Eine Seite nach unten „blättern“.
Control und 0	Entspricht Insert.

Der Zifferntastenblock kann in einen Modus geschaltet werden, in dem diese Tasten auch ohne Control wirksam sind. Die Umschaltung erfolgt über Control+[(NumL) des Ziffernblocks.

Danach ergibt das Drücken einer Taste des Zifferntastenblocks das oben mit Control aufgeführte Kommando. Der NUMLOCK-Modus kann durch erneutes Drücken derselben Tasten wieder aufgehoben werden.

Control-Kommandos

In den anderen Abschnitten wurden viele Control-Kommandos bereits genannt. Hier sollen diese Kommandos und die dort nicht genannten Control-Tasten-Befehle noch einmal zusammengefaßt werden (mit Ausnahme der Control-Kommandos, die mit dem Zifferntastenblock zusammenarbeiten):

Control+Y	Löscht die Zeile, in der der Cursor steht.
Control+U	Fügt diejenige Zeile ein, die als letzte mit Control+Y gelöscht wurde. Dies dient zum einen der Restaurierung von Zeilen, die versehentlich gelöscht wurden. Zum anderen können auf diese Art und Weise sehr leicht einzelne Zeilen verschoben oder kopiert werden (z.B. kopieren: Control-Taste festhalten und dann Y und zweimal U drücken).
Control+N	Fügt eine Leerzeile oberhalb der Befehlszeile ein, in der der Cursor steht (wie Insert), auch dann, wenn in der Zeile Änderungen vorgenommen wurden.
Control+Q	Blockmenü aufrufen (wie Funktionstaste F4).
Control+B	Blockanfang markieren (Block).
Control+K	Blockende markieren (Block).
Control+R	„Blättert" eine Seite nach oben.
Control+C	„Blättert" eine Seite nach unten.
Control+E	Text ersetzen.
Shift+Control+E	Ersetzen mit Abfrage von Such- und Ersatz-String.
Control+F	Text suchen.
Shift+Control+F	Text suchen mit Abfrage des Such-Strings.
Control+Cursor links	Springt an den Anfang der Zeile.
Control+Cursor rechts	Springt an das Ende der Zeile.
Control+Cursor oben	„Blättert" eine Seite nach oben.
Control+Cursor unten	„Blättert" eine Seite nach unten.
Control+Home	Springt an den Anfang des Programms.
Control+Z	Springt an das Ende des Programm-Listings.
Control+Tab	Springt eine Tabulatorposition nach links.
Control+G	Öffnet das Zeilennummernanzeigefeld für die Eingabe einer Zeilennummer (die entsprechende Zeile wird dann angesprungen).

Eine besondere Gruppe von Control-Kommandos ermöglicht das Setzen und Anspringen von Marken im Editor. Diese Marken gelten nur für den Editor, sie haben also nichts mit den Marken zu tun, die von GOTO oder RESTORE benutzt werden. Diese Editor-Marken können an der Cursor-Position durch Drücken von Control und einer Ziffer der Haupttastatur gesetzt werden. Anschließend kann durch Drücken von Alternate und dieser Ziffer an die gewünschte Marke gesprungen werden.

Dabei sind die Tastenkombinationen von Alternate mit den Ziffern 7–9 und 0 vorbelegt. Drücken von Alternate und 7 bewirkt das Springen zur letzten Cursor-Position vor dem Umschalten in den Direkt-Modus oder vor dem letzten Programmstart. Alternate und 8 springt an die Stelle, an der der Cursor beim Start des Editors war. Bei Alternate und 0 wird zur letzten Cursor-Position gesprungen, an der eine Änderung vorgenommen wurde. Alternate und 9 springt an die Stelle, an der der letzte Suchvorgang gestartet wurde.

Mit Hilfe der Delete-Taste kann das Zeichen, auf dem der Cursor steht, gelöscht werden; der Rest der Zeile wird nachgezogen. Die Backspace-Taste löscht das Zeichen, das links vom Cursor steht und zieht den Rest der Zeile nach.

Das Drücken der Tab-Taste (Tabulator) läßt den Cursor an die nächste rechte Tabulatorposition springen. Diese Positionen haben acht Zeichen Abstand. Control+Tab bewegt den Cursor eine Tabulatorposition nach links. Die Return- und die Enter-Taste lassen den Cursor an den Anfang der nächsten Zeile springen. Durch Drücken der Escape-Taste gelangt man in den Direktmodus.

Die Menüleiste und die Funktionstasten

In den obersten beiden Bildschirmzeilen des Editors befinden sich zwei Menüzeilen. Ganz links steht das Amiga-Symbol, bei dessen Anklicken ein Pull-Down-Menü aktiviert wird. Am rechten Rand der Menüleiste befindet sich oben eine Uhr und darunter die Anzeige der aktuellen Textzeilennummer. Der Umgang mit diesen beiden Menüleistenelementen wird am Ende dieses Abschnitts erklärt.

Zwischen dem Amiga-Symbol und der Uhr stehen zwanzig Kommandoworte, jeweils zwei übereinander, die entweder mit der Maus angeklickt oder über die Funktionstasten angesprochen werden können. Die untere Zeile der Kommandoworte läßt sich durch Drücken einer Funktionstaste ansprechen; die Kommandos der oberen Zeile durch Drücken der Shift-Taste und der entsprechenden Funktionstaste.

Zu dem Kommandowort unten links (Load) gehört z.B. die F1-Taste, das Kommandowort darüber (Save) läßt sich mit SHIFT+F1 aufrufen.

Die Menüpunkte haben folgende Bedeutung:

Load (F1)

Das Kommando Load ermöglicht das Laden eines GFA-BASIC-3.0 Programms. Das hier verwendete Format benutzt sogenannte Befehlstokens. Eine solche Programmdatei kann besonders schnell geladen und gespeichert werden. Für dieses Format wird normalerweise die Dateiendung (Extension) .GFA benutzt.

Save (Shift+F1)

Es erscheint der Dateiauswahl-Requester, in der ein Name angegeben werden kann. Unter diesem Namen wird dann das Programm gespeichert, das sich gerade im Editor befindet. Es wird dabei das unter Load genannte tokenisierte Format benutzt. Die voreingestellte Dateikennung (Extension) heißt .GFA. Sie wird, wenn der Benutzer keine Extension angibt, automatisch an den Dateinamen angehängt. Besteht bereits eine Datei unter dem angegebenen Namen, wird diese unter dem gleichen Dateinamen mit der Extension .BAK abgespeichert.

Merge (F2)

Mit diesem Kommando kann eine Datei im ASCII-Format in das aktuelle Programm eingefügt werden. Die Einfügung erfolgt dabei vor der Zeile, in der der Cursor steht. Die voreingestellte Dateikennung ist .LST.
==> Programmzeilen mit dieser Kennung werden beim Mergen eines
==> Programms erzeugt, das der Interpreter nicht versteht.

Save,A (Shift+F2)

Mit Save,A kann das aktuelle Programm im ASCII-Format abgespeichert werden. Eine in diesem Format gespeicherte Datei kann mit Merge wieder geladen werden. Die voreingestellte Dateiendung (Extension) ist .LST. Sie wird, wenn der Benutzer keine Extension angibt, automatisch an den Dateinamen angehängt.

Besteht bereits eine Datei unter dem angegebenen Namen, wird diese unter dem gleichen Dateinamen mit der Extension .BAK abgespeichert.

Llist (F3)

Dieser Befehl leitet den Ausdruck des Programms ein, das sich gerade im Editor befindet. Die Formatierung des Ausdrucks kann durch die sogenannten Punktbefehle beeinflußt werden. Diese Befehle werden wie normale Befehlszeilen in das Programm-Listing eingefügt. Sie lauten (x ist ein Platzhalter für eine Ziffer):

.ll xx	Maximale Zeilenlänge
.pl xx	Maximale Seitenlänge
.ff xxx	Form-Feed-Zeichenkette (für Drucker, die andere Werte für Form-Feed haben, kann ein Ersatz-String angegeben werden. Voreingestellt ist .ff\ 012)
.he kopf	Text der Kopfzeile
.fo fuß	Text der Fußzeile
.lr xx	Linker Rand
.in xxx	Initialisierung, ein String, der einmal am Anfang ausgegeben wird.

.l–	Diese Anweisung bewirkt, daß die nachfolgenden Zeilen nicht mit aufgelistet werden. Das Listing kann durch ein später im Programm folgendes .l+ wieder eingeschaltet werden.
.l+	siehe .l–
.n1 bis .n9	Schaltet Zeilennumerierung mit ein bis neun Stellen ein.
.n0	Schaltet die Zeilennumerierung aus.

In den Text der Kopf- und Fußzeile sowie den FF-String können noch besondere Platzhalter eingefügt werden.

\xxx	Zeichen mit dem ASCII-Code xxx
\d	Datum
\t	Uhrzeit
#	Seitennummer

Um die Sonderzeichen # und \ ausgeben zu können, muß man ein weiteres \-Zeichen davor setzen. Die Zeichenkombination \\ gibt also ein \ auf dem Drucker aus, \# ein #.

Quit (Shift+F3)

Ermöglicht das Verlassen des GFA-BASIC-Interpreters.

Block (F4)

Wenn kein Block markiert ist, und dieser Menüpunkt gewählt wird, so erscheint in der Menüleiste der Text „Block ???“, um anzuzeigen, daß die Wahl des Block-Kommandos in dieser Situation unsinnig ist. Ist aber ein Block markiert, so wird in die oberste Bildschirmzeile ein Menü eingeblendet. Die einzelnen Punkte dieses Menüs können mit der Maus angeklickt werden, sie können aber auch durch Drücken eines Kommandobuchstabens aktiviert werden.

Die Menüpunkte sind (in Klammern steht jeweils die Taste, mit der der Menüpunkt aufgerufen werden kann):

Copy	Kopiert den Block an die aktuelle Cursor-Position. Der Block bleibt dabei markiert.
Move	Verschiebt den Block an die aktuelle Cursor-Position. Die Blockmarkierung wird dabei entfernt.
Write	Speichert den Block als ASCII-Datei ab.
Llist	Druckt den Block aus.
Start	Springt an den Anfang des Blocks.
End	Springt an das Ende des Blocks.
^Del	Löscht den Block (Control-D).
Hide	Entfernt die Blockmarkierung.

Das Drücken einer anderen Taste (oder einer Maustaste) außerhalb dieses Bereiches entfernt das Blockmenü.

New (Shift+F4)

Löscht das Programm, das sich gerade im Editor befindet.

BlkEnd (F5)

Mit BlkEnd kann die Zeile, die vor der Cursor-Zeile steht, als Blockende markiert werden. Befindet sich die Blockanfangmarkierung über dieser Zeile, so wird der Block mit einem Punktraster unterlegt. Dieses Kommando kann auch durch Drücken von Control+K ausgelöst werden.

BlkSta (Shift+F5)

Mit BlkSta wird die Zeile, in der der Cursor steht, als Blockanfang markiert. Ist hinter dieser Zeile bereits eine Blockendemarkierung gesetzt worden, so wird der Blockbereich mit einem Punktraster unterlegt. Dieses Kommando kann auch durch Drücken von Control+B ausgelöst werden.

Find (F6)

Nach Auslösen des Find-Kommandos kann man einen Text angeben, nach dem gesucht werden soll. Die Suche beginnt in der Zeile, in der der Cursor steht. Das Suchen kann mit Control+F oder mit Control+L erneut aufgerufen werden, ohne daß die Abfrage des zu suchenden Textes erscheint.

Wurde der zu suchende Text gefunden, so steht der Cursor am Anfang der Zeile, in der der Suchtext enthalten ist; wurde er nicht gefunden, dann steht der Cursor am Programmende.

Wenn die Find-Anweisung erneut aufgerufen wird, so steht in der Zeile, in der der Such-String eingebenen werden kann, der beim letzten Mal benutzte Such-String. Bei der Eingabe des Suchtextes stehen folgende Editiermöglichkeiten zur Verfügung:

Cursor links	Cursor geht ein Zeichen nach links (sofern er nicht ganz links steht).
Cursor rechts	Cursor geht ein Zeichen nach rechts (sofern er nicht am Ende des Such-Strings steht).
Delete	Löscht das Zeichen des Such-Strings, das unter dem Cursor steht, der Rest des Such-Strings wird von rechts nach links nachgerückt.
Backspace	Löscht das Zeichen links vom Cursor (sofern dieser nicht schon ganz links steht) und zieht den Rest des Such-Strings nach.

Escape	Löscht das Eingabefeld.
Return/Enter	Bestätigt den eingebenen Suchtext und beginnt mit der Suche. Das Find-Kommando kann auch durch die Tastenkombination Shift+Control+F oder Shift+Control +L aufgerufen werden. In „zusammengeklappten" (siehe Help-Taste im Abschnitt Cursor-Tasten-Block) Prozeduren wird kein Text gefunden.

Replac (Shift+F6)

Dieses Kommando dient dem Ersetzen eines Textes durch einen anderen. Nach seinem Aufruf wird zunächst nach dem Text gefragt, der ersetzt werden soll. Anschließend wird nach dem Text gefragt, mit dem zu ersetzen ist. Nun wird im Programm ab der Cursor-Position mit der Suche nach dem zu ersetzenden Text begonnen. Wenn der Suchtext gefunden wird, so springt der Cursor an den Anfang der entsprechenden Zeile. Nun kann die Ersetzung durch Drücken der Tastenkombination Control+E vorgenommen werden. Weiteres Betätigen dieser Tastenkombination führt dazu, daß die nächste ersetzbare Textstelle aufgesucht wird oder, falls sie sich in der aktuellen Cursor-Zeile befindet, ersetzt wird.

Das Replac-Kommando kann auch durch die Tastenkombination Shift+-Control+E aufgerufen werden. Die Editiermöglichkeiten, die bei der Eingabe des Such- und des Ersatztextes zur Verfügung stehen, wurden bei der Beschreibung des Find-Befehls aufgeführt. In „zusammengeklappten" (siehe Help-Taste im Abschnitt Cursor-Tasten-Block) Prozeduren wird kein Text gefunden.

Pg Down (F7)

„Blättert" den Programmtext um eine Bildschirmseite nach unten. Kann auch durch Control+C aufgerufen werden.

Pg Up (Shift+F7)

„Blättert" den Programmtext um eine Bildschirmseite nach oben. Kann auch durch Control+R aufgerufen werden.

Insert/Overwr (F8)

Schaltet zwischen dem Einfüge- und dem Überschreibmodus hin und her.

Interl/Normal (Shift+F8)

Im Modus „Normal" wird der Editor in einem Screen mit der Auflösung „Hires" angezeigt. In dieser Auflösung können 28 Zeilen dargestellt werden. Im Modus „Interl" wird der Screen in der Auflösung „Hires-Lace" dargestellt, was auf normalen Monitoren ein leichtes Flackern des Bildes verursacht. In diesem Modus können 57 Zeilen angezeigt werden. Mit Interl/Normal kann zwischen diesen beiden Modi hin- und hergeschaltet werden.

ClkOn/ClkOff (F9)

Mit diesem Kommando kann die Uhranzeige an oder abeschaltet werden.

Direct (Shift+F9)

Dieses Kommando ruft den Direktmodus auf. In diesem Modus können GFA-BASIC-Befehle eingegeben werden, die dann nach Drücken der Return- oder Enter-Taste sofort abgearbeitet werden. Einige Befehle, z.B. Schleifenbefehle, sind im Direktmodus allerdings nicht aufrufbar. Diesen Modus kann man auch durch Drücken der Escape-Taste aufrufen.

Außerdem ist es im Direktmodus möglich, mit den Tasten „Cursor hoch" und „Cursor runter" die letzten acht Befehle (die im Direktmodus eingegeben wurden) wiederzuholen.

Drücken von Alt+Help bewirkt die Anzeige des jeweils letzten Befehls. Mit den Tastenkombinationen Control+Cursor links bzw. Control+Cursor rechts lassen sich der Zeilenanfang und das Zeilenende erreichen. Zusätzlich ist es durch Drücken der Insert-Taste möglich, zwischen Einfüge- und Überschreibmodus hin- und herzuschalten.

Durch Drücken der Escape-Taste, gefolgt von Return, kann man den Direktmodus wieder verlassen. Alternativ dazu ist das Verlassen auch durch das gleichzeitige Drücken von Control+Shift+Alternate möglich.

Um mehrzeilige Anweisungen aufzurufen, kann man im Editor eine Prozedur schreiben, die diese Befehle beinhaltet. Im Direktmodus kann diese Prozedur dann aufgerufen werden.

Test (F10)

Nach der Auslösung dieser Anweisung wird geprüft, ob alle Schleifen, Unterprogramme und bedingte Anweisungen des aktuellen Programms geschlossen sind. Es wird also eine Strukturprüfung vorgenommen. Mit Hilfe von Control, linker Shift-Taste und Alternate kann ein laufendes Programm unterbrochen werden.

Run (Shift+F10)

Startet das Programm, das sich gerade im Editor befindet. Enthält dieses Programm einen strukturellen Fehler, z.B. eine nicht abgeschlossene FOR-NEXT-Schleife, so wird eine entsprechende Fehlermeldung ausgegeben und das Programm nicht gestartet.

Control+Shift+Alternate

Mit Hilfe der Tastenkombination Control+linke Shift-Taste+Alternate kann das laufende Programm unterbrochen werden.

Die Zeilenanzeige und die Uhr

Rechts unten in der Menüleiste befindet sich eine Anzeige, in der die Nummer der Zeile steht, in der sich der Cursor gerade befindet. In diese Zeilenanzeige kann auch eine Zeilennummer eingetragen werden, so daß der Cursor in die entsprechende Zeile springt. Zu diesem Zweck kann das Anzeigefeld entweder mit der Maus angeklickt oder durch das Kommando Control+G geöffnet werden.

Es werden in diesem Feld nur Ziffern als Eingabe akzeptiert. Die Editiermöglichkeiten für den Eintrag der gewünschten Zeilennummern sind:

Cursor links	Cursor geht ein Zeichen nach links (sofern er nicht ganz links steht).
Cursor rechts	Cursor geht ein Zeichen nach rechts (sofern er nicht am Ende des Eingabefeldes steht).
Backspace	Hat dieselbe Funktion wie Cursor links.
Escape	Löscht das Eingabefeld.
Return/Enter	Bestätigt die eingegebene Zahl.

Über der Zeilennummernanzeige ist eine Uhr, die die Systemzeit anzeigt. Nach dem Anklicken der Uhr mit dem Mauspfeil hat man die Möglichkeit, die Uhr neu zu stellen. Die Editiermöglichkeiten für die Eingabe der Uhrzeit sind dieselben wie bei der Eingabe ins Zeilennummernfeld. Lediglich die Escape-Taste hat hier eine andere Bedeutung, da sie die Eingabe abbricht, wobei der alte Wert für die Uhrzeit erhalten bleibt.

Im GFA-BASIC-Editor ist ein kleines Menü implementiert, das, wie üblich, über die rechte Maustaste oder auch über die rechte Maustaste und ein weiteres Zeichen erreicht werden kann. Dieses Menü kann durch Anklicken des großen Amiga-A-Symbols aus- und eingeschaltet werden.

Load (Amiga-L) wie F1

Save (Amiga-S) wie Shift-F1

Neue Namen (Amiga-N) schaltet eine Nachfrage-Funktion für neue Variablennamen ein oder aus. Wenn eingeschaltet, erfolgt bei der Einführung eines neuen Variablennamens eine Rückfrage, die, im Falle eines Tippfehlers, mit Error beantwortet werden kann. Im Direktmodus erfolgt diese Rückfrage immer, weil durch neue Variablen die alten gelöscht werden.

Run (Amiga-R) wie Shift-F10

Taskpri 0 (Amiga-0) ändert die Taskpriorität auf 0, sodaß Hintergrundtasks auch Rechenzeit bekommen.

Taskpri 1 (Amiga-1) ändert die Taskpriorität auf 1. So wird trotz weiterer rechenintensiver Programme dieses Programm schnell abgearbeitet. Also erfolgt das Scrollen in diesem Editor mit annehmbarer Geschwindigkeit, obwohl z. B. ein anderes GFA-BASIC-Programm eine umfangreiche Rechnung durchführt, ohne zwischenzeitlich auf Tastatureingaben zu warten.
Mittels ~SetTaskPri(FindTask(0),n) kann die Taskpriorität weiterhin geändert (etwa im Rahmen -4 bis +4) werden.

Cleanup (Amiga-C) bricht alle laufenden Töne ab, stoppt alle Bobs und gibt alle Sprites frei.

NewCLI Dieser Menüpunkt erlaubt das Starten eines neuen CLI.

Geänderte Editoreigenschaften im GFA-BASIC 3.5

1. Der Editor benötigt bei jeder Programmzeile zwei Bytes mehr als der alte Editor (Vs. 3.0-3.041). Dadurch wird das "Zurück-Scrollen" schneller und das Falten auch von Funktionen möglich.

2. Beim "Suchen" kann auch in den Kopfzeilen geschlossener Prozeduren bzw. Funktionen gesucht werden.

3. Beim Listen werden "Labels" um zwei Zeichen nach links ausgerückt (wie bei CASE).

Tab-Funktionen

Tab	Cursor springt auf die nächste Tabulatorposition.
Ctrl + Tab	Cursor springt auf die letzte vorhergehende Tabulatorposition.
Shift-links + Tab	Fügt Leerzeichen bis zur nächsten Tabulatorposition ein.
Shift-rechts + Tab	Löscht alle Leerzeichen eine Zeile bis und ab Cursor.

2 – Variablen und Speicherverwaltung

Variablentypen

In GFA-BASIC 3.0 beginnen Variablennamen mit einem Buchstaben (A bis Z, À bis ß) gefolgt von weiteren Buchstaben, Ziffern (0-9), Punkten(.) oder Unterstrichen(_).

GFA-BASIC 3.0 verfügt über folgende Variablentypen:

Name	Postfix	Speicherplatzbedarf
Boolean	!	1 Byte (in Arrays ein Bit)
Byte	|	1 Byte
Word	&	2 Byte
Integer	%	4 Byte
Float	#	8 Byte
String	$	(Abhängig von der String-Länge)

Bool'sche Variablen können nur die Werte 0 (FALSE) oder –1 (TRUE) annehmen. Wird ihnen ein Wert zugewiesen, der ungleich Null ist, so wird dieser Wert als –1 aufgefaßt. Dieser Variablentyp wird mit dem Postfix ! versehen und belegt 1 Byte Speicherplatz. In bool'schen Arrays benötigt ein Arrayelement lediglich ein Bit.

Beispiele:

```
b!=TRUE
c!=x>y
```

Der Variablentyp Byte kann Werte zwischen 0 und 255 annehmen. Größere Werte für x werden als Überlauf behandelt. Das Postfix dieses Variablentyps ist der senkrechte Strich |. Wie der Name schon sagt, belegt dieser Variablentyp ein Byte.

```
x|=128
```

Word ist ein vorzeichenbehafteter 2-Byte-Integertyp. Das Postfix dieses Typs ist &. Der darstellbare Zahlenbereich reicht von –32768 bis 32767.

x&=32767

Integer ist ein vorzeichenbehafteter 4-Byte-Integertyp. Als Postfix wird das Zeichen % benutzt. Der darstellbare Zahlenbereich reicht von –2147483648 bis 2147483647.

x%=2000000000

Float ist ein Fließkommavariablentyp mit 8 Bytes Speicherplatzbedarf. Bei diesem Typ wird kein Postfix oder das #-Zeichen verwendet. Der darstellbare Zahlenbereich reicht von 2.225073858507E –308 bis 3.595386269725E +308.

Zeichenketten (Strings) werden mit dem Postfix $ gekennzeichnet. Sie können eine maximale Länge von 32767 Zeichen haben. Strings werden mit Hilfe eines sogenannten Deskriptors verwaltet. Ein solcher Deskriptor hat eine Länge von sechs Bytes. Die ersten vier Bytes enthalten die Adresse der Zeichenkette, die letzten beiden Bytes die Länge des Strings. Hinter der Zeichenkette wird ein Füllbyte eingefügt, falls die Länge des Strings ungerade ist. Danach folgt die Adresse des Deskriptors (Backtrailer).

Die Adressen aller Variablen-Typen können mit Hilfe der Funktionen VARPTR (oder V:) und ARRPTR (oder *) ermittelt werden. Bei Strings erhält man mit VARPTR die Adresse des ersten Bytes der Zeichenkette, mit ARRPTR erhält man die Adresse des String-Deskriptors. Bei Arrays kann man mit VARPTR / V: die Adressen der einzelnen Array-Elemente ermitteln (z.B. V: x%(5)). Mit Hilfe von ARRPTR / * erhält man die Adresse des Array-Deskriptors (z.B. ARRPTR (x%())). Mit VARPTR / V: kann man also die Adresse einzelner Feldelemente ermitteln, während ARRPTR die Adresse des Feld-Deskriptors bestimmt.

Felder

Von allen Variablentypen lassen sich Felder erzeugen. Das Einrichten eines solchen Feldes erfolgt mit Hilfe des DIM-Befehls. Die Nachfragefunktion DIM? gibt an, wie viele Elemente ein Array enthält.

Die Verwaltung von Arrays im Speicher erfolgt über Deskriptoren. Ein solcher Deskriptor ist eine sechs Bytes lange Struktur, in deren ersten vier Bytes die Adresse des Feldes enthalten ist. Die nächsten beiden Bytes geben an, wie viele Dimensionen das Feld hat. Das Feld beginnt mit jeweils vier Bytes für jede Arraydimension, wobei mit der letzten Dimension begonnen wird. Dahinter folgen dann die Inhalte der Feldelemente. Bei String-Variablen sind dies die Deskriptoren der Zeichenketten.

Beispielsweise ergibt sich nach DIM a%(2,3) folgende Situation: mit Hilfe von *a%() erhält man die Adresse des Felddeskriptors. Die Anzahl der Dimensionen des Arrays steht in den letzten beiden der sechs Deskriptor-Bytes, also ergibt INT{*a%()+4} den Wert 2. Das Array beginnt mit der Anzahl der Unterteilungen der zweiten Dimension, wobei das nullte Element mitgezählt wird (vorausgesetzt, OPTION BASE 0 gilt). Dahinter steht die Anzahl der Unterteilungen der ersten Dimension. Dann ergibt {{*a%()}} den Wert 4 und {{*a%()}+4} den Wert 3.

Daran schließen sich die 12 Array-Elemente an, mit jeweils 4 Bytes Speicherplatzbedarf in der Reihenfolge:

a%(0,0) a%(1,0) a%(2,0) a%(0,1) a%(1,1) a%(2,1) usw.

DEFBIT f$
DEFBYT f$
DEFINT f$
DEFWRD f$
DEFFLT f$
DEFSTR f$

f$: String-Konstante

Die Anweisung DEFxxx dient der vereinfachten Variablendeklaration. Dabei ist xxx ein Platzhalter für die Abkürzungen von Variablentypen, die in der folgenden Tabelle aufgeführt und erläutert werden:

DEFxxx	f$	Postfix	Alle Variablen...
DEFBIT	„b“	!	mit dem Anfangsbuchstaben „b“ werden als Bool'sche Variablen deklariert.
DEFBYT	„by“	|	mit beiden Anfangsbuchstaben „b“ und „y“ werden als 1-Byte-Integer deklariert.
DEFWRD	„w“	&	mit dem Anfangsbuchstaben „w“ werden als 2-Byte-Integer mitVorzeichen deklariert.
DEFINT	„i–k,m–p“	%	mit den Anfangsbuchstaben „i“ bis „k“ und „m“ bis „p“ werden als 4-Byte-Integer mit Vorzeichen deklariert.
DEFFLT	„x–z“	#	mit den Anfangsbuchstaben „x“ bis „z“ werden als 8-Byte-Fließkommawert deklariert.
DEFSTR	„s,t“	$	mit dem Anfangsbuchstaben „s“ und „t“ werden als Zeichenkette deklariert.

Die Angabe von DEFSNG oder DEFDBL wird vom Editor automatisch durch DEFFLT ersetzt. Es ist normalerweise sinnvoll, solche globalen Deklarationen zu Programmbeginn durchzuführen. Soll an irgendeiner Stelle des Programms von der mit DEFxxx festgelegten Definition abgewichen werden, muß nur das gewünschte Postfix hinter der Variable angegeben werden. Diese explizite Angabe des Variablentyps hat immer Vorrang vor den globalen Definitionen. Die Deklaration von Variablentypen ist solange gültig, bis eine andere Deklaration vorgenommen wird. Der voreingestellte Variablentyp entspricht DEFFLT.

LET x=y

x: avar oder svar
y: aexp oder sexp

Mit Hilfe von LET kann man einer Variable den Wert eines Ausdrucks zuweisen.

Der Ausdruck und die Variable müssen dabei entweder beide numerisch oder beide vom Zeichenkettentyp sein. Normalerweise ist die Benutzung der LET-Anweisung nicht notwendig. Sie diente in älteren BASIC-Dialekten dazu, Befehlsworte als Variablen verwenden zu können. GFA-BASIC erkennt aber gewöhnlich automatisch, ob ein Befehlswort als Variable verwendet wird.

Beispiel:

```
LET print=3
PRINT print
```

CLEAR
CLR x [,y,...]

x, y: svar oder avar

Mit CLEAR werden alle Variablen und Felder gelöscht. Der Befehl kann nicht innerhalb von FOR-NEXT-Schleifen oder Unterprogrammen benutzt werden. Beim Starten eines Programms wird er automatisch ausgeführt, und er ist sonst nur zur Behandlung schwerwiegender Fehler mit RESUME x sinnvoll anwendbar.

Der Befehl CLR löscht die Variablen, die in der hinter ihm stehenden Liste aufgeführt sind. Arrays können mit CLR allerdings nicht gelöscht werden.

Beispiel:

```
x=2
y=3
CLEAR
PRINT x,y
'
x=2
y=3
CLR x
PRINT x,y
```

Schreibt dreimal die Zahl 0 und einmal die Zahl 3 auf den Bildschirm.

ERASE z1() [,z2(),...]

z1, z2: Name beliebiger Arrays

ERASE löscht komplette Arrays, die nach diesem Kommando auch erneut dimensioniert werden können. Mit dem ERASE-Befehl können auch mehrere Arrays gelöscht werden, zum Beispiel ERASE x(),y().

Beispiel:

```
DIM x(10)
PRINT FRE(0)
ERASE x()
PRINT FRE(0)
```

Es werden zwei Zahlen ausgegeben, die zeigen, daß der durch DIM belegte Speicherplatz durch ERASE wieder freigegeben wird.

DIM x(d1,[d2,...]) [,y(d1,[d2,...])]
DIM?(x())

x, y: Variablenname (beliebiger Variablentyp)
d1, d2: iexp

Mit Hilfe des Befehls DIM können numerische und Zeichenkettenfelder eingerichtet werden. Der Aufbau eines solchen Arrays wurde in der Einführung dieses Abschnitts beschrieben.

Aus praktischer Sicht ist die mögliche Anzahl der Dimensionen des Arrays nicht begrenzt. Die Anzahl der Array-Elemente bei mehrdimensionalen Arrays ist insofern beschränkt, als daß die letzte Dimension kleiner als 65535 sein muß. Ebenso muß das Produkt aus der Anzahl der Feldelemente kleiner als 65535 sein. (So ist z.B. DIM a%(100,10,10) erlaubt, da die letzte Dimension (10) und das Produkt aus der Anzahl der Feldelemente (100*10*10 = 10000) kleiner als 65535 sind.)

Mit Hilfe von OPTION BASE kann festgelegt werden, ob es ein nulltes Array-Element gibt. In einem Array können nur Variablen desselben Typs existieren. Das Ansprechen der Arrayelemente erfolgt über deren Indizes. Die Funktion DIM? ermittelt, wie viele Elemente das Array hat.

Beispiel:

```
DIM x(10)
x(4)=3
PRINT x(LEN("Test"))
PRINT DIM?(x())
DIM y%(2,3)
PRINT DIM?(y%())
```

Richtet zwei Arrays ein. Schreibt in eines der Arrays einen Wert und liest ihn wieder aus. Schreibt die Anzahl der Elemente der beiden Arrays auf den Bildschirm (11 und 12).

OPTION BASE 0
OPTION BASE 1

Mit Hilfe des Befehls OPTION BASE kann bestimmt werden, ob bei Arrays ein nulltes Element existieren soll oder nicht. Mit OPTION BASE 0 kann ein solches nulltes Element eingerichtet werden, mit OPTION BASE 1 wird es verhindert. In diesem Fall beginnen die Array-Indizes bei 1.

Der Inhalt von Feldern wird durch den OPTION BASE-Befehl nicht verändert, allerdings ändert sich unter Umständen der Index der einzelnen Elemente um den Wert 1.

Wird kein OPTION BASE-Befehl gegeben, so gibt es Array-Elemente mit dem Index 0.

Beispiel:

```
DIM x%(3)
FOR i%=3 DOWNTO 0
  x%(i%)=i%
  PRINT i%,x%(i%)
NEXT i%
OPTION BASE 1
FOR i%=3 DOWNTO 0
  PRINT i%,x%(i%)
NEXT i%
```

Schreibt die Indizes und den Inhalt des Arrays x%() auf den Bildschirm. Das Programm endet mit der Fehlermeldung „Feld-Index falsch", da x%() nach OPTION BASE 1 kein nulltes Element mehr besitzt.

INSERT x(i)=y
DELETE x(i)

x: Name eines Arrays
i: iexp
y: aexp oder sexp, je nach Variablentyp des Arrays

Mit den Befehlen INSERT und DELETE kann ein Element in einem Array eingefügt oder gelöscht werden. INSERT fügt in das Array x an die Position i den Wert des Ausdrucks y ein. Dabei werden alle Elemente des Arrays, die einen Index größer i haben, um eine Position nach „oben" verschoben. Wenn ein Element z.B. vorher an der Position i+3 stand, so steht es nach dem INSERT-Befehl an der Position i+4. Das letzte Element des Arrays wird bei diesem Einfügevorgang gelöscht.

DELETE entfernt das i-te Element des Arrays x. Alle Array-Elemente, die einen Index größer i haben, werden dabei eine Position nach „unten" verschoben. Das letzte Element des Arrays wird dabei gleich 0 (bei Zeichenkettenfeldern zu einem Leer-String). Diese beiden Befehle eignen sich vor allem zur Verwaltung von Listen, in die ständig Elemente eingefügt und gelöscht werden müssen.

Beispiele:

```
DIM x%(5)
FOR i%=1 TO 5
  x%(i%)=i%
NEXT i%
INSERT x%(3)=33
FOR i%=0 TO 5
  PRINT x%(i%)
NEXT i%
```

```
DIM x%(5)
FOR i%=1 TO 5
  x%(i%)=i%
NEXT i%
DELETE x%(3)
FOR i%=0 TO 5
  PRINT x%(i%)
NEXT i%
```

Das linke Beispiel gibt die Zahlen 0, 1, 2, 33, 3 und 4 aus. Rechts werden die Zahlen 0, 1, 2, 4, 5 und 0 ausgegeben.

ARRAYFILL x(),y

x: Name eines Arrays mit numerischem Variablentyp
y: aexp

Der Befehl ARRAYFILL setzt alle Elemente des Arrays x dem Wert des numerischen Ausdrucks y gleich. Der Wert y darf den Wertebereich der Array-Elemente nicht überschreiten. Das heißt beispielsweise, einem Array das aus Byte-Einträgen besteht, kann kein Integer-Wert zugewiesen werden.

Beispiel:

```
DIM x&(10)
PRINT x&(4)
ARRAYFILL x&(),5+1
PRINT x&(4)
```

Gibt die Zahl Null aus, da beim DIM-Befehl automatisch alle Feldelemente auf Null gesetzt werden. Nach Füllen des Arrays wird die Zahl 6 auf den Bildschirm ausgegeben. Wäre bei ARRAYFILL anstatt 5+1 die Zahl 70000 angegeben, welche zu groß ist, würde GFA-BASIC die Meldung „Zahl nicht Wort“ ausgeben.

ASC(a$)
CHR$(x)

a$: sexp
x: aexp

ASC und CHR$ sind Umkehrfunktionen zueinander. Die Funktion ASC liefert den ASCII-Code des ersten Zeichens im String a$. Ist a$ ein Leer-String, so wird eine Null zurückgemeldet. CHR$ liefert das ASCII-Zeichen, das den Code x hat. Dabei wird nur das Lowbyte von x bewertet (entspricht dem Ausdruck x AND 255).

Beispiel:

```
PRINT ASC("TEST")
code|=ASC(CHR$(65))   ! CHR$(65) ergibt „A“
PRINT code|
```

Schreibt die Zahlen 84 und 65 auf den Bildschirm.

BIN$(x [,y])
OCT$(x [,y])
HEX$(x [,y])

x, y: iexp

BIN$ wandelt eine Integerzahl in die binäre Darstellung. Das bedeutet, daß die Zahl nach der Umwandlung im Zahlensystem zur Basis 2 vorliegt. Dabei werden Zahlen nur durch die Ziffern 0 und 1 dargestellt. Der Parameter „y" gibt an, wie viele Stellen (1 bis 32) verwendet werden.

OCT$ wandelt einen Integerwert in die entsprechende oktale Darstellung im Zahlensystem zur Basis 8. Dabei werden die Zahlen durch die Ziffern 0 bis 7 dargestellt. Der optionale Parameter „y" gibt an, wie viele Stellen (1 bis 11) angegeben werden.

HEX$ wandelt eine Integerzahl in die entsprechende sedezimale (oder hexadezimale) Darstellung um. Dabei handelt es sich um ein Zahlensystem zur Basis 16 mit den Ziffern 0 bis 9 und den Buchstaben A bis F. Der wahlweise zweite Parameter „y" gibt an, wie viele hexadezimale Ziffern dargestellt werden (maximal acht).

HEX$(x,n) entspricht RIGHT$(STRING$(n,48)+HEX$(x),n).

Beispiel:

```
x=32+15
a$=OCT$(16+7,4)
PRINT HEX$(x),a$,BIN$(1+4+16+64,8)
```

Schreibt 2F, 0027 und 01010101 auf den Bildschirm.

STR$(x)
STR$(x,y)
STR$(x,y,z)

x, y, z: aexp

STR$ wandelt eine Zahl „x“ in einen String. Ein zweiter Parameter „y“ gibt an, auf wieviele Stellen der Ergebnis-String links mit Leerzeichen aufgefüllt wird bzw. wie viele Zeichen des Ergebnis-Strings (vom Zeichenkettenende gezählt) verwendet werden.

STR$(x,n) entspricht RIGHT$(SPACE$(n)+STR$(x),n).

Eine weitere Variante von STR$ verfügt über einen dritten Parameter „z“. Dabei wird die Zahl „x“ auf „y“ Stellen mit „z“ Nachkommastellen formatiert und gerundet.

Beispiel:

```
a=123.4567
PRINT STR$(a,6,2)
PRINT STR$(PI,5,3)
PRINT STR$(PI,2,2)
```

Schreibt die Zahlen 123.46, 3.142 und 14 auf den Bildschirm.

VAL(a$)
VAL?(a$)

a$: sexp

VAL() wandelt eine Zeichenkette in eine Zahl. Trifft VAL() dabei auf ein Zeichen, das nicht mehr als Bestandteil einer Zahl interpretiert werden kann, so wird die Analyse des Strings abgebrochen. Die am String-Anfang stehende Zahl wird dann von VAL zurückgemeldet. Ist am String-Anfang keine Zahl zu finden, ergibt VAL Null.

Durch Angabe von &H(hex) oder &X(bin) oder &O(oct) können Zahlen in hexadezimaler, binärer und oktaler Schreibweise erkannt werden.

Außerdem können durch $ hex-Zahlen und durch % bin-Zahlen gekennzeichnet werden.

Mit VAL?() kann man überprüfen, wie viele Stellen einer Zahl mit VAL() umgewandelt werden. VAL?() ergibt Null, wenn eine ungültige Zahl angegeben wird.

Beispiele:

```
a$=STR$(12345)
PRINT VAL(a$),VAL("-.123 abc 123"),VAL?("3.00 DM")
```

Gibt die Zahlen 12345, –.123 und 4 auf dem Bildschirm aus.

```
PRINT VAL("&H"+"AF")
```

Gibt die Zahl 175 aus.

```
PRINT VAL("$AA")
PRINT VAL("%10101010")
```

Gibt zweimal die Zahl 170 aus.

CINT(x)
CFLOAT(y)

x: aexp
y: iexp

Die Funktion CINT wandelt eine Fließkommazahl „x" in einen gerundeten Integerwert. Analog wandelt CFLOAT einen Integerwert „y" in eine Fließkommazahl. Diese Funktion wird normalerweise nicht benötigt und ist nur der Vollständigkeit halber aufgeführt. Im Compiler hat sie jedoch eine Bedeutung.

Beispiel:

```
a=1.2345
a%=10000
b%=CINT(a)
b=CFLOAT(a%)
PRINT b%,b
```

Gibt 1 und 10000 auf dem Bildschirm aus.

CVI(a$)
CVL(a$)
CVS(a$)
CVD(a$)
MKI$(x)
MKL$(x)
MKS$(x)
MKD$(x)

a$: sexp
x: aexp

Die Funktionen CVI, CVL, CVS, CVF und CVD wandeln Zeichenketten in Zahlen um.

Im Gegensatz zu VAL/STR$ erfolgt keine Umwandlung in Klartext, sondern die interne Darstellung des Strings und der Zahlen ist entscheidend. Die einzelnen CVx-Funktionen haben folgende Wirkungen:

CVI wandelt eine 2-Byte-Zeichenkette in eine 16-Bit-Integerzahl.

CVL wandelt eine 4-Byte-Zeichenkette in eine 32-Bit-Integerzahl.

CVS wandelt eine 4-Byte-Zeichenkette, die eine Zahl im Amiga-BASIC kompatiblen Format enthält, in eine GFA-BASIC-Fließkommazahl um.

CVD wandelt eine 8 Byte Zeichenkette in eine Fließkommazahl im GFA-BASIC 3.0-Format um.

MKI$, MKL$, MKS$ und MKD$ sind die Umkehrfunktionen der CVx-Funktionen; ihre Bedeutung ergibt sich also aus der gerade aufgeführten Tabelle.

Dabei entspricht:

```
MKI$(x%)=CHR$(SHR(x%,8))+CHR$(x%)
MKL$(x%)=CHR$(SHR(x%,24))+CHR$(SHR(x%,16))+CHR$(SHR(x%,8))+CHR$(x%)
```

Das heißt, daß das höchstwertige Byte zuerst kommt.

Ein Anwendungsbeispiel ist das Lesen von Zahlenformaten anderer Programme sowie das platzsparende Speichern von Zahlen zum Beispiel in„R"-Files.

Beispiel:

```
a$=MKL$(1000)
PRINT CVL(a$),LEN(a$)
b$=MKD$(100.1)
PRINT CVD(b$),LEN(b$)
```

Schreibt die Zahlen 1000 und 4 sowie die Zahlen 100.1 und 8 auf den Bildschirm.

SWAP

SWAP a,b
SWAP e(),f()
SWAP *c,d()

a, b: avar oder svar
c: Zeiger auf einen Array-Deskriptor
d, e, f: Name von Arrays

In seiner einfachsten Variante dient der Befehl SWAP dem Vertauschen von zwei Variablen desselben Typs (SWAP a,b). Er kann aber auch zum Vertauschen von zwei Feldern verwendet werden. Das Feldvertauschen geht sehr schnell, da nur die zugehörigen Deskriptoren vertauscht werden. So wird auch die Dimensionierung der beiden Arrays vertauscht. Arrays müssen hierzu nicht dimensioniert sein.

In der dritten Variante enthält c den Pointer auf einen Array-Deskriptor und d() ist der Name eines Arrays. Dies ist vor allem bei der indirekten Übergabe von Feldern an Unterprogramme interessant (siehe zweites Beispiel).

Der Befehl SWAP sollte von der Funktion SWAP, die im Abschnitt über Bit-Operationen besprochen wird, klar unterschieden werden.

Beispiele:

```
x=1
y=2
PRINT x,y
SWAP x,y
PRINT x,y
```

Gibt nach 1 und 2 auch die Zahlen 2 und 1 aus.

```
DIM x(3)
change(*x())
PRINT x(2)
'
PROCEDURE change(adr%)
  SWAP *adr%,a()
  ARRAYFILL a(),1
  SWAP *adr%,a()
RETURN
```

Das Array x() wird in der Prozedur change mit Einsen gefüllt, ohne daß der Name x() im Unterprogramm auftaucht. So kann ein Unterprogramm für mehrere Felder verwendet werden. Dann wird, wie im Beispiel, die Deskriptoradresse übergeben und durch SWAP am Anang und am Ende werden diese Felder dann unter einem Namen angesprochen.

Es empfiehlt sich die indirekte Feldübergabe als VAR-Parameter:

```
DIM x(3)
CHANGE (x())
PRINT x(2)
'
PROCEDURE change (VAR a())
  ARRAYFILL a(),1
RETURN
```

QSORT a(s) [,n] [,j%()]
QSORT x$(s) OFFSET o WITH i() [,n [,j%()]]
SSORT a(s) [,n] [,j%()]
SSORT x$(s) OFFSET o WITH i() [,n [,j%()]]

a():	beliebiges Array, auch String-Array
i():	Integer-Array (|,& oder %)
j%():	4-Byte-Integer-Array
x$():	String-Array
n, o:	iexp
s:	+, – oder kein Zeichen

Mit Hilfe der Befehle SSORT und QSORT können die Elemente eines Arrays nach ihrer Größe sortiert werden. Bei SSORT wird dazu das Shellsort-Verfahren benutzt, bei QSORT das Quicksort-Verfahren.

In die Klammern, vor denen der Name des zu sortierenden Arrays steht, kann ein Plus- oder Minus-Symbol eingesetzt werden. Das Minus-Zeichen besagt, daß die Elemente des Arrays in absteigender Reihenfolge sortiert werden sollen. In diesem Fall steht das größte Element nach der Sortierung im nullten Array-Element. Das Plus-Zeichen führt dazu, daß das Array in aufsteigender Reihenfolge sortiert wird; im nullten Array-Element steht dann nach der Sortierung der kleinste Wert. Ohne Symbol wird aufsteigend sortiert (wie +).

Der Parameter „n“ gibt an, daß nur die ersten „n“ Elemente des Arrays sortiert werden sollen (falls OPTION BASE 0 gilt, sind dies die Elemente mit den Indizes 0 bis n–1, sonst 1 bis n). Durch „o“ kann ein OFFSET festgelegt werden. Dies ist die Zahl der Zeichen, die ab dem Beginn des Strings übersprungen werden sollen. Diese Zeichen werden also beim Sortieren nicht beachtet. Ist ein String kürzer als die Angabe in „o“, wird ein Leerstring als Sortierkriterium benutzt.

Als dritter Parameter kann ein weiteres Integer-Array angegeben werden, das bei der Sortierung des ersten Arrays mitsortiert wird. Jede Vertauschung von Elementen im ersten Array wird auch im zweiten durchgeführt. Dies kann z.B. verwendet werden, wenn in einem Array ein Sortierschlüssel (z.B. Postleitzahl) und in mehreren Arrays weitere Informationen hierzu enthalten sind. Das mitsortierte Integerfeld enthält dabei z.B. Array-Indizes, „R“-Datei-Indizes, LOC-Werte für SEEK usw.

Bei der Sortierung von String-Arrays kann mit WITH ein Sortierkriterium in Form eines Arrays mit mindestens 256 Elementen angegeben werden. Ohne Angabe von WITH wird die normale ASCII-Tabelle als Sortierkriterium verwendet (siehe zweites Beispiel).

Beispiele:

```
DIM x%(20)
PRINT "Unsortiert:                ";
FOR i%=0 TO 10
  x%(i%)=RAND(9)+1
  PRINT x%(i%);"  ";
NEXT i%
PRINT
'
QSORT x%(),11
DIM index%(20)
PRINT "Absteigend sortiert:  ";
FOR i%=0 TO 10
  PRINT x%(i%);"  ";
  index%(i%)=i%
NEXT i%
PRINT
'
SSORT x%(-),11,index%()
PRINT "Aufsteigend sortiert: ";
FOR i%=0 TO 10
  PRINT x%(i%);"  ";
NEXT i%
PRINT
PRINT "Mitsortiertes Feld:   ";
FOR i%=0 TO 10
  PRINT index%(i%);"  ";
NEXT i%
```

Gibt ein unsortiertes Feld und zwei sortierte Reihen von Zufallszahlen aus. In einer vierten Reihe stehen die Werte eines mitsortierten Feldes.

```
DIM b|(256)
FOR i%=0 TO 255
  b|(i%)=ASC(UPPER$(CHR$(i%)))
NEXT%
FOR i%=1 TO 7
  READ a$,b$
  b|(ASC(a$))=ASC(b$)
NEXT i%
DATA Ä,A,Ö,O,Ü,U,ä,A,ö,O,ü,U,ß,S
DIM n$(3)
FOR i%=0 TO 3
  READ n$(i%)
NEXT i%
DATA Muhlmann,Müller,Überlauf,Unterlauf
QSORT n$(),4
FOR i%=0 TO 3
  PRINT n$(i%),
NEXT i%
PRINT
QSORT n$() WITH b|(),4
FOR i%=0 TO 3
  PRINT n$(i%)
NEXT i%
```

Sortiert zweimal das String-Array n$(), einmal ohne WITH, einmal mit. Die Angabe von WITH b|() führt hier dazu, daß die deutschen Umlaute wie die entsprechenden Nicht-Umlaute und die Kleinbuchstaben bei Großbuchstaben einsortiert werden.

FALSE
TRUE
PI

Die beiden Konstanten FALSE und TRUE enthalten Werte für logisch falsch (0) und logisch wahr (–1). Die Konstante PI enthält den Wert der Kreiszahl Pi.

Beispiel:

```
PRINT FALSE
IF TRUE
  PRINT PI
ENDIF
```

Schreibt die Zahlen 0 und 3.14159265359 auf den Bildschirm.

DATE$
TIME$

zeit$, datum$: sexp

Die Funktion DATE$ ermittelt das Systemdatum im Format: TT.MM.JJJJ (Tag.Monat.Jahr) oder MM/TT/JJJJ (US-Format, siehe MODE). TIME$ ermittelt die Systemuhrzeit. Hier ist das Format: HH:MM:SS (Stunden: Minuten: Sekunden). Mit DATE$ und TIME$ können Datum und Uhrzeit auch gesetzt werden. Dabei wird an DATE$ das Datum als String im entsprechenden Format zugewiesen. Gleiches gilt für TIME$.

Beispiel:

```
PRINT DATE$,TIME$
DATE$="27.2.1988"
TIME$="20:15:30"
PRINT DATE$,TIME$
```

Schreibt das aktuelle Systemdatum und die Systemzeit auf den Bildschirm, setzt diese Variablen neu und gibt sie danach wieder aus.

SETTIME zeit$,datum$

zeit$, datum$: sexp

Mit dem Befehl SETTIME können Uhrzeit und Datum gestellt werden. Dabei müssen die Strings zeit$ und datum$ das Format haben, das gerade bei TIME$ und DATE$ angegeben wurde. Lediglich bei Jahresangaben zwischen 1978 und 2077 kann die Jahrhundertangabe entfallen. Wird SETTIME mit einem falschen Format der Strings zeit$ oder datum$ benutzt, so werden die aktuellen Werte nicht verändert.

Beispiel:

```
PRINT DATE$,TIME$
SETTIME "20:15:30","27.2.1988"
PRINT DATE$,TIME$
```

Schreibt das aktuelle Systemdatum und die Systemzeit auf den Bildschirm, setzt diese Variablen neu und gibt sie danach wieder aus.

TIMER

Mit der Funktion TIMER erhält man die Zeit seit Mitternacht in zweihundertstel Sekunden. Die Auflösung dieses Timers beträgt aber nur etwa eine Zehntel Sekunde.

Beispiel:

```
t%=TIMER
FOR i%=1 TO 25000
NEXT i%
PRINT (TIMER-t%)/200
```

Gibt die Zeit in Sekunden an, die für die Leerschleife benötigt wird.

FRE()
FRE(x)

x: aexp

Diese Funktion berechnet den freien Speicherplatz. Der Parameter x bleibt dabei unberücksichtigt. Darüber hinaus löst FRE(x) eine Garbage-Collection aus (Sammeln der noch benötigten String-Bereiche am oberen Speicherende). FRE() ergibt die Größe des Speicherbereichs, der ohne diese Garbage-Collection noch frei ist.

Beispiel:

```
frei%=FRE(0)
max%=frei%/3/4
DIM x%(max%)
PRINT frei%,max%
```

Dimensioniert ein Array so, daß es etwa ein Drittel des freien Speicherplatzes belegt. Ein 4-Byte-Integer-Feld belegt 4 Bytes pro Element, deshalb die Division durch 4.

TYPE(x)

x: iexp

Mit Hilfe der Funktion TYPE kann überprüft werden, ob ein korrekter Pointer auf eine Variable übergeben wurde. Es wird eine Kennzahl für den Typ dieser Variable oder der Wert –1 zurückgegeben. Man erhält, je nach Variablentyp, auf den x zeigt, folgende Werte:

Float	→	0
String	→	1
Integer	→	2
Boolean	→	3
Float-Array	→	4
String-Array	→	5
Integer-Array	→	6
Bool'sches Array	→	7
Word	→	8
Byte	→	9
Word-Array	→	12
Byte-Array	→	13

Beispiel:

```
a$="test"
x%=4
DIM y(3)
PRINT TYPE(*a$),TYPE(*x%),TYPE(*y())
```

Schreibt die Zahlen 1, 2 und 4 auf den Bildschirm.

DUMP [a$ [TO b$]]

a$, b$: sexp

Mit Hilfe des DUMP-Befehls können Variableninhalte während des Programmlaufs ausgegeben oder Labels, Prozeduren und Funktionen aufgelistet werden. Dazu kann der String-Ausdruck a$ folgende Werte annehmen:

Beispiele:

DUMP

Gibt alle Variablenwerte und die Dimensionierungen von Arrays aus.

DUMP "a"

Wie oben, aber nur Variablen bzw. Arrays, die mit „a" beginnnen.

DUMP ":"

Listet alle Labels (Marken) und gibt dahinter die Editor-Zeilennummer an, wo die Marke definiert wird. Die Variation (:b) listet nur Marken, die mit „b" beginnen.

DUMP "@"

Listet alle Prozeduren und Funktionen auf, wobei hinter den Namen die Zeilennummer steht, an der diese im Editor zu finden sind:

proc_name @ 100	(Prozedur)
func_name FN 200	(Funktion mit numerischer Rückgabe)
func_name $ FN 300	(Funktion mit String-Rückgabe)

Nicht mehr definierte Labels, Prozeduren und Funktionen werden ohne Zeilennummer angegeben. Wird das Programm-Listing unter der Option SAVE,A abgespeichert und neu eingeladen, erscheinen diese undefinierten Namen nicht mehr. Es erscheinen aber Label-, Prozedur- und Funktionsnamen ohne Zeilennummern, die verwendet werden, aber nicht definiert sind.

Zu den hinter den Namen angegebenen Zeilennummern kann im Editor mit Control+G gesprungen werden.

Bei der Ausgabe von Strings werden maximal 60 Zeichen dargestellt. Ist die Zeichenkette länger, wird als letztes Zeichen ein „ >“ ausgegeben, sonst ein Hochkomma (”). Steuerzeichen (d.h. ASCII-Wert < 32) werden durch einen Punkt dargestellt.

Die oben genannten Ausgaben können auch in eine Datei umgeleitet werden. Dazu muß in b$ ein Dateiname angegeben werden.

Als Default-Extension (ohne Punkt) wird .DMP angenommen.

VARPTR(x)
V:x
ARRPTR(y)
***y**

x: Name einer Variablen beliebigen Typs
y: Name einer Variablen oder eines Arrays mit leeren Klammern

Mit ARRPTR(y) oder *y wird die Adresse einer Variablen y, bei Arrays oder Strings die Adresse des Deskriptors, bestimmt.

Mit VARPTR(x) oder V:x wird dagegen bei Strings die Adresse des Strings selber oder bei Arrays die Adresse eines Array-Elementes bestimmt.

Bei skalaren (nicht Feld oder String) Variablen sind VARPTR, V:, ARRPTR und * gleichbedeutend.

Beispiel:

```
DIM x%(10)
a$="Test"
PRINT ARRPTR(x%())),VARPTR(x%(0)),V:x%(1)
PRINT ARRPTR(a$), *a$,VARPTR(a$)
```

Gibt in der ersten Zeile die Adresse des Array-Deskriptors sowie die Adresse der ersten beiden Feldelemente von x%() aus. In der zweiten Zeile wird die Adresse des String-Deskriptors von a$ zweimal und die Adresse des ersten Bytes der Zeichenkette einmal ausgegeben.

*x

x: svar oder ein Array-Name gefolgt von ()
Das Multiplikationszeichen kann auch als Pointersymbol (Zeigersymbol) dienen.

In diesem Fall ergibt *x die Adresse, an der die Variable x im Speicher steht. Bei Zeichenketten ergibt *x$ die Adresse des String-Descriptors (ARRPTR (x$)). Der Ausdruck *x ist gleichbedeutend mit ARRPTR(x). Dies gilt auch für Arrays.

Eine besondere Bedeutung hat diese Variante der Pointeranwendung bei der indirekten Übergabe von Arrays und Variablen an Unterprogramme. Dazu kann man besser die Anweisung VAR benutzen.

Beispiel:

```
' indirekte Array-Übergabe
DIM a(3)
change(*a())
PRINT a(2)
'
PROCEDURE change(ptr%)
  SWAP *ptr%,x()
  ARRAYFILL x(),1
  SWAP *ptr%,x()
RETURN
```

Der Inhalt des Arrays a() wird verändert, ohne daß sein Name in der Prozedur change auftaucht. Es wird die Zahl 1 auf den Bildschirm geschrieben (siehe auch SWAP). Das gleiche Beispiel mit VAR:

```
DIM a(3)
change(a())
PRINT a(2)
'
PROCEDURE change(VAR x())
  ARRAYFILL x(),1
RETURN
```

POKE x,y
DPOKE x,y
LPOKE x,y

x, y: iexp

Mit Hilfe des Befehls POKE und dessen Varianten kann man Werte in den Speicher schreiben. Die einzelnen Befehlsvarianten sind:

POKE x,y Schreibt den Wert y als ein Byte an die Adresse x.

DPOKE x,y Schreibt y als 2-Byte-Wert an die Adresse x.

LPOKE x,y Schreibt y als 4-Byte-Wert an die Adresse x.

Bei der Verwendung von DPOKE und LPOKE muß darauf geachtet werden, daß nur gerade Adressen angegeben werden.

Beispiel:

```
DATA 45,20,30,72,84,213,55,153,90,12,145,43
'
IF adr%=MALLOC(24,2)  ! reserviert 24 Byte Speicher im Chip-Ram
  '
  FOR i&=0 TO 23
    READ b&
    POKE adr%+i&,b&
  NEXT i&
ENDIF
```

Reserviert Speicher, liest die Byte-Werte der DATA-Zeile aus und schreibt sie in den Speicher.

PEEK(x)
DPEEK(x)
LPEEK(x)

Mit den PEEK-Funktionen wird der Inhalt von Speicherstellen ausgelesen:

PEEK(x)	Liest ein Byte an der Adresse x.
DPEEK(x)	Liest zwei Bytes ab der Adresse x.
LPEEK(x)	Liest vier Bytes ab der Adresse x.

Wie bei den POKE-Befehlen muß auch bei den PEEK-Funktionen darauf geachtet werden, daß nur gerade Adressen angegeben werden.

Beispiel:

```
FOR i&=0 TO 23
  PRINT PEEK adr%+i&
NEXT i&
```

Gibt die mit PEEK ausgelesenen Werte auf dem Bildschirm aus.

BYTE{x}
CARD{x}
INT{x}, WORD{x}
LONG{x}, {x}
FLOAT{x}
SINGLE{x}
DOUBLE{x}
CHAR{x}

x: iexp

Mit Hilfe dieser Kommandos kann man bestimmte Variablentypen ab einer Adresse lesen oder an eine Adresse schreiben. Als Funktion (z.B. y= BYTE{x}) wird ab der Adresse x aus dem Speicher gelesen. Als Befehl (z.B. BYTE{x}=y) wird ab der Adresse x der Wert y in den Speicher geschrieben.

Bei der Verwendung von INT{}, CARD{}, LONG{}, {}, FLOAT{}, SINGLE{} und DOUBLE{} muß darauf geachtet werden, daß nur gerade Adressen angegeben werden, da sonst ein Adreßfehler ausgelöst wird.

Typ	Bedeutung
BYTE{x}	Liest/schreibt ein Byte.
CARD{x}	Liest/schreibt 2-Byte-Integer ohne Vorzeichen.
INT{x}, WORD{x}	Liest/schreibt 2-Byte-Integer mit Vorzeichen.
LONG{x}, {x}	Liest/schreibt 4-Byte-Integerwert.
FLOAT{x}	Liest/schreibt 8-Byte-Fließkommavariable im GFA-BASIC 3.0-Format.
SINGLE{x}	Liest/schreibt 4-Byte-Fließkommavariable im IEEE-Single-Format.
DOUBLE{x}	Liest/schreibt eine 8-Byte-Fließkommavariable im IEEE-Double-Format.
CHAR{x}	Liest/schreibt einen mit einem Null-Byte endenden String. (Besonders wichtig für die Kommunikation mit C-Routinen!)

Mit x%=LONG{adr%} wird der Variablen x% das Langwort, das an der Adresse adr% steht, zugewiesen und bei LONG{adr%}=x% wird an die Adresse adr% der Wert der Variablen x% als Langwort geschrieben.

Mit den SINGLE- und DOUBLE-Anweisungen ist es möglich, ein fremdes Zahlenformat zu lesen bzw. zu schreiben. Einige C-Compiler kennen das IEEE-DOUBLE-Format. Um z.B. eine GFA-BASIC-Zahl in das SINGLE- oder DOUBLE-Format zu konvertieren und so hexadezimal auszugeben, kann man folgendes schreiben:

```
a$=SPACE$(4)
SINGLE{V:a$}=1.2345
PRINT HEX$(CVL(a$),8)
```

oder

```
a$=SPACE$(8)
DOUBLE{V:a$}=1.2345
PRINT HEX$({V:a$},8)
PRINT HEX$({V:a$+4},8)
```

Einige der oben genannten Funktionen entsprechen in etwa den PEEK-Funktionen, LONG{x} (oder {x}) entspricht etwa LPEEK(x).

Beispiel:

```
a$=CHAR{{SCREEN(1)+22}}
```

Liest den Titel eines Screens aus.

ABSOLUTE x,y

x: eine Variable beliebigen Typs
y: iexp

Mit dem Befehl ABSOLUTE wird die Adresse (ARRPTR/*) einer Variablen verändert. Die Variable wird ab sofort an der angegebenen Adresse verwaltet und abgelegt. Dabei Sie nimmt auch den Wert des Inhalts der neuen Adresse an. Es darf in y kein ungerader Wert angegeben werden, sofern x keine Boolean-, Byte- oder String-Variable ist.

Dieser Befehl sollte nur von systemerfahrenen Programmierern genutzt werden. Die Zuweisung einer Adresse kann nicht beliebig vorgenommen, ohne da dies Auswirkungen auf das gesamte Amiga-System hat. Mit ABSOLUTE können aber beispielsweise festen Systemadressen Variablennamen zugewiesen werden.

Beispiel:

```
x=13
y=7
ABSOLUTE y,*x
PRINT x, y, *x, *y
```

Hier wird die Variable y auf die Adresse von x gelegt, so daß am Ende beide Variablen mit gleichem Wert (7) und gleicher Adresse ausgeben werden.

VOID fx
~fi

fx: aexp
fi: iexp

In Programmiersprachen unterscheidet man normalerweise Befehle und Funktionen. Befehle führen dazu, daß irgendeine Tätigkeit durchgeführt wird.

Bei Funktionen ermittelt diese Tätigkeit einen Wert, der dann zurückgemeldet wird. Dieser Rückgabewert kann dann Element eines Ausdrucks sein, mit PRINT ausgegeben werden oder auch mit einem Gleichheitszeichen einer Variablen zugewiesen werden, usw.

In vielen Fällen ist der Programmierer aber nicht an diesem Rückgabewert interessiert, sondern nur an der von der Funktion ausgeübten Tätigkeit. So meldet z.B. die Funktion WAIT ein Signal-Bit zurück. Soll das Programm aber lediglich warten, so ist das Signal-Bit nicht von Bedeutung. In einem solchen Fall kann GFA-BASIC mit VOID angewiesen werden, die Funktion auszuführen, aber den Rückgabewert zu vergessen. Im Gegensatz zu VOID wird bei Verwendung der Tilde (~) ein Integerausdruck berechnet und dann vergessen. Bei VOID wird z.B. bei WAIT() noch eine (überflüssige) Umwandlung in eine Fließkommazahl durchgeführt. Der Compiler erkennt diese überflüssigen Berechnungen selbst.

Beispiel:

```
VOID WAIT()
```

oder

```
~WAIT()
```

Wartet auf eine Nachricht ohne die, hier überflüssige, Zuweisung dummy=WAIT(). Mit der Tilde spart der Interpreter auch noch die Umwandlung des Ergebnisses von WAIT() in einen Fließkommawert.

MALLOC(x,art)
MFREE(y,a)

x, y, art, a: iexp

Die Funktion MALLOC dient der Reservierung (Allokierung) von Speicherbereichen.

Ist x eine positive Zahl, so bedeutet dies, daß x Bytes reserviert werden sollen. In diesem Fall gibt MALLOC die Anfangsadresse des reservierten Speicherbereichs zurück. Trat beim Reservierungsversuch ein Fehler auf, so wird 0 zurückgemeldet.

Im Ausdruck art wird angegeben welche Art Speicher allokiert werden soll. Art kann folgende Werte annehmen.

2 (&H2) MEMF_CHIP
Chip Memory soll geholt werden. Speicherbereich, der auch von den Grafik-Chips genutzt werden kann.

1 (&H1) MEMF_PUBLIC
Diese Art sollte gewählt werden, wenn die Kompatibilität des Programmes zu späteren Betriebssytemversionen sicher gestellt werden soll.

4 (&H4) MEMF_FAST
Es soll Speicherbereich außerhalb des Grafik-Chip Speichers geholt werden.

65536 (&H10000) MEMF_CLEAR
MEMF_CLEAR kann mit allen anderen Arten kombiniert werden. Bevor MALLOC die Speicheradresse zurückmeldet, wird der Speicherbereich mit Nullen aufgefüllt.

Der durch Malloc angeforderte Speicherbereich liegt immer außerhalb des vom Interpreter belegten Variablenspeichers.

Allokierte Speicherbereiche sollten unbedingt vor der Beendigung des Programms wieder freigegeben werden. Dies wird beim Verlassen des Interpreters nicht automatisch durchgeführt.

MFREE (FreeMem) gibt den mit MALLOC reservierten Speicherplatz wieder frei. Dabei steht in y die Anfangsadresse des freizugebenden Speicherblocks, also ein bei MALLOC erhaltener Rückgabewert. a gibt an, wieviel Bytes freigegeben werden sollen.

Beispiel:

```
adr%=MALLOC(60000,&H10000)
PRINT adr%
IF adr%>0
  rueck%=MFREE(adr%,60000)
ENDIF
```

Reserviert einen Speicherbereich von 60000 Bytes, gibt die Adresse auf dem Bildschirm aus und macht ihn (wenn erfolgreich reserviert) wieder frei.

BMOVE ab_adr,an_adr,anzahl

ab_adr, an_adr, anzahl: iexp

Mit Hilfe des Befehls BMOVE können Speicherbereiche kopiert werden. Im Ausdruck ab_adr steht, ab welcher Adresse der zu kopierende Bereich steht. In an_adr steht, an welche Adresse die Kopie des Speicherbereichs abgelegt werden soll. In anzahl steht die Anzahl der zu kopierenden Bytes.

Der Befehl wird bei geraden Parametern deutlich schneller abgearbeitet als bei ungeraden. Er kann auch verwendet werden, wenn sich Orginal- und Kopierbereich überlagern.

Beispiel:

```
OPENS 1,0,0,320,200,1,128
BM%=LPEEK(SCREEN(1)+88)
BP%=LPEEK(BM%+16)
DIM A%(1000)
BMOVE BP%, *A%(),2000
```

Kopiert die ersten 1000 Bytes aus der Screenbitplane in das Array A%.

RESERVE [n]

n: iexp

Mit Hilfe des RESERVE-Befehls kann angegeben werden, wie groß der von GFA-BASIC benutzte Speicherbereich sein soll. Beim Start von GFA-BASIC ist ein Bereich von 64 KByte verfügbar. In „n" wird angegeben, welcher Wert der Befehl FRE(0) anschließend zurückliefert.

Bei der Ausführung dieses Befehls werden alle Variablen gelöscht. Deshalb sollte RESERVE nur am Anfang eines Programmes benutzt werden.

Beispiel:

```
RESERVE 100000
PRINT FRE(0))
```

Reserviert 100000 Bytes für das Programm und gibt die Programm-Speichergröße aus.

3 – Operatoren

Operatorenhierarchie

Wenn sich verschiedene Operatoren in einem Ausdruck befinden, so werden sie in einer bestimmten Reihenfolge abgearbeitet. Diese Reihenfolge ist abhängig von ihrem Platz in der sogenannten Operatorenhierarchie. Die Operatoren, die in dieser Hierarchie ganz oben stehen, werden als erste bearbeitet.

Die Hierarchie lautet:

( )	Klammern
+	Zeichenketten-Addition
= < > => <= <>	Zeichenketten-Vergleich
+ –	Vorzeichen
^	Potenzierung
* /	Multiplikation, Division
DIV MOD	Ganzzahlige Division und Modulo
+ –	Addition, Subtraktion
= == < <= >= > <>	Vergleichsoperatoren
AND OR XOR IMP EQV	Logische Operatoren
NOT	Negation

Mit Hilfe der Klammern ist es möglich, daß Operatoren, die in der Hierarchie niedrig stehen, dennoch zuerst abgearbeitet werden. In der Rangfolge der Operatoren ist auch die bekannte Regel „Punkt- vor Strichrechnung" zu erkennen.

Beispiel:

```
PRINT 2+4*3
PRINT (2+4)*3
PRINT 2+(4*3)
PRINT 3 *2^2
```

Auf dem Bildschirm erscheinen die Zahlen 14, 18, 14 und 12.

Logische Operatoren

Die logischen Operatoren AND, OR, XOR, NOT, IMP und EQV arbeiten auf Bit-Ebene für 32-Bit-Integerwerte.

Logische Operatoren verknüpfen zwei logische Ausdrücke und erzeugen einen Wahrheitswert (wahr oder falsch). Eine Ausnahme bildet der Operator NOT, der den Wahrheitswert des hinter ihm stehenden logischen Ausdrucks negiert.

Der numerische Wert für logisch falsch (FALSE) ist 0. Jeder andere Zahlenwert wird als logisch wahr interpretiert. Die Konstante TRUE hat dabei den Wert –1. Alle logischen Operatoren können auch auf numerische Ausdrücke angewendet werden. In diesem Fall werden die logischen Operationen bitweise ausgeführt.

Die Wirkung logischer Operatoren läßt sich am besten mit sogenannten Wahrheitstabellen beschreiben. In diesen Tabellen stehen in den ersten Spalten die Wahrheitswerte der verknüpften Ausdrücke (Operanden) und in der letzten Spalte der erzeugte logische Wert.

NOT x

x: iexp

Der Operator NOT (Negation) negiert den hinter ihm stehenden logischen Ausdruck. Er ist der einzige logische Operator, der nur ein Argument hat.

x	NOT x
w	f
f	w

Beispiele:

```
PRINT NOT FALSE
PRINT NOT TRUE
PRINT NOT 0
```

Es erscheinen die Zahlen –1, 0 und –1 auf dem Bildschirm.

```
x=1
PRINT BIN$(x,2)
PRINT BIN$(NOT x,2)
```

01 und 10 erscheint auf dem Bildschirm.

```
x%=17
PRINT BIN$(x%,8),x%
PRINT BIN$(NOT x%,8),NOT x%
```

Ausgabe:
00010001 17
11101110 –18

x AND y

x, y: iexp

Der logische Operator AND (Konjunktion) prüft, ob zwei logische Ausdrücke x und y beide wahr sind. Nur in diesem Fall erzeugt er den Wert TRUE (logisch wahr, –1). Ist einer der beiden oder sind beide logischen Ausdrücke falsch, so erzeugt auch AND ein logisches falsch.

x	y	x AND y
w	w	w
w	f	f
f	w	f
f	f	f

Beispiele:

```
PRINT TRUE AND –1
PRINT FALSE AND TRUE
```

Auf dem Bildschirm erscheinen die Zahlen –1 und 0.

```
x=3
y=10
PRINT BIN$(x,4)
PRINT BIN$(y,4)
PRINT BIN$(x AND y,4),x AND y
```

Ausgabe:
0011
1010
0010
2

x OR y

x, y: iexp

Der Operator OR (Disjunktion) prüft, ob mindestens einer von zwei logischen Ausdrücken x und y wahr ist. In diesem Fall wird logisch wahr erzeugt. Nur wenn x und y beide logisch falsch sind, ergibt auch x OR y logisch falsch. Im Unterschied zu XOR wird für den Fall, daß x und y beide wahr sind, auch ein logisches wahr erzeugt.

OR bedeutet, daß eines oder beide Argumente wahr sind.

x	y	x OR y
w	w	w
w	f	w
f	w	w
f	f	f

Beispiele:

```
PRINT TRUE OR -1
PRINT FALSE OR TRUE
PRINT 0 OR FALSE
```

Es erscheinen die Zahlen –1, –1 und 0 auf dem Bildschirm.

```
x=3
y=10
PRINT BIN$(x,4)
PRINT BIN$(y,4)
PRINT BIN$(x OR y,4),x OR y
```

0011
1010
1011 und die Zahl 11 erscheinen auf dem Bildschirm.

x XOR y

x, y: iexp

Dieser Operator prüft, ob einer von zwei logischen Ausdrücken x und y logisch wahr ist. In diesem Fall ergibt auch x XOR y logisch wahr. Sind x und y beide logisch wahr oder beide logisch falsch, so ergibt XOR logisch falsch.

Der Unterschied zu OR besteht darin, daß TRUE OR TRUE wahr und TRUE XOR TRUE falsch ist (XOR = exklusive Disjunktion).

XOR ergibt dann (und nur dann) wahr, wenn in genau einem seiner Argumente das entsprechende Bit gesetzt ist.

x	y	x XOR y
w	w	f
w	f	w
f	w	w
f	f	f

Beispiele:

```
PRINT FALSE XOR -1
PRINT -1 XOR 1
PRINT 0 XOR FALSE
```

Es erscheinen die Zahlen –1, 0 und 0 auf dem Bildschirm.

```
x=3
y=10
PRINT BIN$(x,4)
PRINT BIN$(y,4)
PRINT BIN$(x XOR y,4),x XOR y
```

Ausgabe:
0011 1010 1001 9

x IMP y

x, y: iexp

Der Operator IMP (Implikation) entspricht einer logischen Folgerung. Eine solche logische Folgerung kann nur dann falsch gewesen sein, wenn aus einer wahren Aussage etwas Falsches folgt. Dementsprechend ist x IMP y nur dann falsch, wenn x wahr und y falsch ist.

Im Gegensatz zu AND, OR, XOR und EQV ist hier die Reihenfolge der Argumente wichtig.

x	y	x IMP y
w	w	w
w	f	f
f	w	w
f	f	w

Beispiele:

```
PRINT TRUE IMP -1
PRINT 0 IMP FALSE
PRINT TRUE IMP 0
```

Es erscheinen die Zahlen –1, –1 und 0 auf dem Bildschirm.

```
x=3
y=10
PRINT BIN$(x,4)
PRINT BIN$(y,4)
PRINT BIN$(x IMP y,4)
```

Ausgabe:
0011
1010
1110

x EQV y

x, y: iexp

Der Operator EQV (Äquivalenz) erzeugt dann logisch wahr, wenn die logischen Ausdrücke x und y den gleichen Wahrheitswert haben.

EQV setzt die Bits, die in beiden Argumenten gleich sind. Dies ist genau das Gegenteil von XOR, so daß (x EQV y) gleichbedeutend ist mit (NOT x XOR y).

x	y	x EQV y
w	w	w
w	f	f
f	w	f
f	f	w

Beispiel:

```
PRINT TRUE EQV FALSE
PRINT FALSE EQV FALSE
```

Auf dem Bildschirm erscheinen die Zahlen 0 und –1.

```
x=3
y=10
PRINT BIN$(x,4)
PRINT BIN$(y,4)
PRINT BIN$(x EQV y,4)
```

Ausgabe:
0011
1010
0110

x + y
x – y
a$+ b$

x, y: aexp
a$, b$: sexp

Mit + und – werden beliebige arithmetische Ausdrücke addiert und subtrahiert. Plus und Minus werden darüberhinaus als Vorzeichen verwendet. Bitte beachten Sie dabei allerdings die unterschiedliche Operatorenhierarchie (siehe Kapitelanfang).

Mit dem Operator + können auch Zeichenketten verknüpft werden. Das Ergebnis ist eine Zeichenkette, die a$ und b$ enthält. Eine Minus-Verknüpfung ist für Zeichenketten nicht möglich.

Beispiel:

```
a$="GFA–"
PRINT a$+"BASIC"
```

Auf dem Bildschirm erscheint der Text „GFA-BASIC".

x= y

x, y: exp

Der Operator = vergleicht zwei numerische oder Zeichenkettenausdrücke auf Gleichheit. Wenn die beiden Ausdrücke gleich sind, so wird logisch wahr (–1) erzeugt, andernfalls logisch falsch (0).

Beispiel:

```
x=6
IF 2=x/3
  PRINT "Ok"
ENDIF
PRINT 2=x/3
```

Es erscheint Ok und –1 auf dem Bildschirm.

x= y

x: var
y: exp

Das Gleichheitszeichen = kann nicht nur als Vergleichsoperator benutzt werden, sondern auch als Zuweisungsoperator. Dabei wird der Wert des rechts vom Gleichheitszeichen stehenden Ausdrucks y bestimmt und der links stehenden Variable x zugewiesen.

Numerische Ausdrücke können nur numerischen Variablen zugewiesen werden, Zeichenkettenausdrücke können nur String-Ausdrücken zugewiesen werden.

Zur Zuweisung gibt es auch noch den gleichwertigen Befehl LET var=exp (vgl. LET), der auch die Zuweisung an Variablen wie z.B. let oder rem erlaubt.

Beispiel:

```
x=LEN("TEST")+3
a$="GF"+CHR$(65)
PRINT x,a$
```

Auf dem Bildschirm erscheint 7 und „GFA".

x== y

x, y: aexp

Vergleicht zwei numerische Ausdrücke auf ungefähre Gleichheit. Bei diesem Vergleich werden achteinhalb Nachkommastellen (28 Bit der Mantisse der Fließkommazahl) für den Vergleich herangezogen. Die Verwendung von == ist dann sinnvoll, wenn Fließkommazahlen verwendet werden, bei denen Rundungsungenauigkeiten auftreten können.

Dieser Befehl kann nicht in Zusammenhang mit Zeichenketten benutzt werden.

Beispiele:

```
PRINT 1.0000000001=1
PRINT 1.0000000001==1
```

Es erscheinen die Zahlen 0 und –1 auf dem Bildschirm.

```
a=SINQ(77)
b=SIN(RAD(77))
PRINT a=b
PRINT a==b
```

Es erscheinen die Zahlen 0 (logisch falsch) und –1 (logisch wahr).

x< y
x> y
x< = y
x> = y

x, y: exp

Diese Operatoren dienen dem Größenvergleich von numerischen und String-Ausdrücken. Bei numerischen Ausdrücken werden die Zahlen verglichen, die sich aus ihnen berechnen lassen.

Bei String-Ausdrücken richtet sich der Vergleich nach dem ASCII-Code der Zeichen. Der String „ABC“ wird als die Zahlfolge 65, 66 und 67 behandelt. Lautet die Aussage "ABC">"AAA", so werden zunächst die ersten beiden Zeichen verglichen, die beide den ASCII-Code 65 haben. Nun wird das nächste Zeichen zum Vergleich herangezogen. Da B (ASCII-Code 66) einen höheren Code als A hat, wird B als „größer“ aufgefaßt. An dieser Stelle wird der Vergleich der beiden Strings abgebrochen, und das Ergebnis der Aussage ist logisch wahr, „ABC“ ist also größer als „AAA“.

Ein Sonderfall des String-Vergleichs tritt ein, wenn einer der beiden Strings endet, bevor ungleiche Zeichen entdeckt wurden. Ein Beispiel wäre: "AA">"A". Diese Aussage ist logisch wahr, da ein nicht existierendes Zeichen als „kleinstes Zeichen“ gewertet wird. Selbst "A"+Chr$(0)>"A" ist logisch wahr. Nun zu den einzelnen Operatoren:

x>y ist wahr, wenn x größer als y ist.
x<y ist wahr, wenn x kleiner als y ist.
x>=y ist wahr, wenn x größer oder gleich y ist.
x<=y ist wahr, wenn x kleiner oder gleich y ist.

Die Schreibweisen x<=y und x=<y sowie x>=y und x=>y sind gleichwertig. Auch wird x><y in x<>y umgewandelt.

Beispiel:

```
PRINT "AAA">"aaa"
PRINT -1<=4-5
```

Es erscheinen die Zahlen 0 und –1 auf dem Bildschirm.

x< > y

x, y: exp

Dieser Operator prüft, ob zwei numerische oder String-Ausdrücke ungleich sind. Wenn dies der Fall ist, so ist die Aussage x<>y logisch wahr. Sind x und y gleich, so wird x<>y logisch falsch. Die Schreibweisen x<>y und x><y sind gleichwertig.

Beispiel:

```
PRINT "Test"<>"test"
PRINT -1<>4-5
```

Es erscheinen die Zahlen –1 und 0 auf dem Bildschirm.

4 – Numerische Funktionen und Zeichenketten

ADD x,y
SUB x,y
MUL x,y
DIV x,y

x: avar
y: aexp

Der Befehl ADD erhöht die Variable x um den Wert y, während SUB x um den Wert von y vermindert. MUL multipliziert x mit y und weist das Ergebnis x zu. DIV teilt die Variable x durch y und legt das Resultat in x ab.

Bei diesen Befehlen muß x eine numerische Variable sein, während y ein numerischer Ausdruck ist. Diese Befehle arbeiten auch mit Fließkomma-Variablen, sind jedoch mit Integer-Variablen erheblich schneller. Wichtig ist, daß ADD, SUB, MUL und DIV bei Integervariablen (%,&,|) keine Überlaufprüfung durchführen.

Der Befehl MUL a%,3.5 wird wie MUL a%,3 behandelt, da bei Integer-Variablen auch eine Integer-Multiplikation bzw. -division stattfindet.

ADD x,y	entspricht	x=x+y
SUB x,y	entspricht	x=x–y
MUL x,y	entspricht	x=x*y
DIV x,y	entspricht	x=x/y

Beispiel:

```
x%=1
y%=2
z%=3
ADD x%,y%                !nun ist x gleich 3
SUB z%,(x%-1)/2          !der numerische Ausdruck (x%-1)/2 ergibt 1
PRINT x%,y%
```

Auf dem Bildschirm erscheinen die Zahlen 3 und 2.

ADD(x,y)
SUB(x,y)
MUL(x,y)
DIV(x,y)
MOD(x,y)

x, y: iexp

Diese Funktionen können an Stelle folgender numerischer Operatoren verwendet werden:

ADD(x,y)	entspricht	x+y
SUB(x,y)	entspricht	x–y
MUL(x,y)	entspricht	x*y
DIV(x,y)	entspricht	x\y (x DIV y)
MOD(x,y)	entspricht	x MOD y

Da die o.g. Funktionen mit Integerarithmetik arbeiten, werden Nachkommastellen ignoriert. Nach den Anweisungen

```
x%=5
y%=4
ADD y%,3
z%=SUB(x%,3)
```

Hat x% den Wert 5, y% den Wert 7 und z% den Wert 2.

Die Funktionen ADD, SUB, MUL, DIV und MOD können beliebig verschachtelt werden. Diese Reihenfolge von Operatoren und Operanden nennt man Polnische Notation.

Beispiele:

```
DEFINT "a-z"
x=4
y=ADD(x,x)     !y wird gleich 8
z=SUB(x,2)
PRINT y,z,ADD(x,MUL(y,2))
```

Auf dem Bildschirm erscheinen die Zahlen 8, 2 und 20.

```
DEFINT "a-z"
x=2
y=MUL(x,3)                    !y wird gleich 6
PRINT y,DIV(8,x),MOD(11,4)    !MOD(11,4) ist gleich 3
```

Auf dem Bildschirm erscheinen die Zahlen 6, 4 und 3.

```
DEFINT "a-z"
x=5
y=ADD(SUB(x,2),MUL(3,4))
PRINT y,DIV(8,MOD(14,4))
```

Auf dem Bildschirm erscheinen die Zahlen 15 und 4.

SIN(x)
COS(x)
TAN(x)
ASIN(x)
ACOS(x)
ATN(x)

x, grad: aexp

Dies sind die trigonometrischen Funktionen. Der numerische Ausdruck x wird als Bogenmaß interpretiert. Die Funktionen berechnen:

SIN	Sinus
COS	Cosinus
TAN	Tangens
ASIN	Arcussinus
ACOS	Arcuscosinus
ATN	Arcustangens

Beispiel:

```
x=90
y=COS(x *PI/180)
z=270 *PI/180
PRINT  y,SIN(z),TAN(45),ATN(1/2)
```

Es erscheinen die Zahlen 1,–1,1.619775190544 und 0.4636476090008.

SINQ(grad)
COSQ(grad)

grad: aexp

Die Funktionen SINQ und COSQ liefern interpolierte Sinus- bzw. Cosinuswerte, dazu wird GFA-BASIC-intern eine Tabelle mit den Sinuswerten in Grad-Schritten benutzt. Entsprechend dem Wert des Funktionsarguments grad werden die Zwischenwerte in 1/16-Grad-Schritten linear interpoliert.

Für die grafische Darstellung auf dem Bildschirm ist diese Genauigkeit ausreichend und von den mit SIN bzw. COS berechneten Werten nicht zu unterscheiden, aber wesentlich schneller (etwa um den Faktor 10). Im Unterschied zu SIN und COS erwarten SINQ und COSQ ihre Parameter im Gradmaß.

SINQ(grad)	entspricht	SIN(RAD(grad))
COSQ(grad)	entspricht	COS(RAD(grad))

Beispiele:

```
alpha%=30
PRINT SINQ(alpha%)
```

Gibt 0.5 aus.

```
bogen=PI/2
langsam=SIN(bogen)
schnell=SINQ(DEG(bogen))
PRINT langsam schnell
```

Berechnet jeweils den gleichen Wert. Beachten Sie die Umwandlung ins Gradmaß bei SINQ.

DEG(x)
RAD(grad)

x, grad: aexp

Um einen Winkel vom Bogen- ins Gradmaß umzurechnen, benutzt man die Funktion DEG() und deren Umkehrfunktion RAD(). DEG(x) entspricht dabei (x*180/PI).

SQR(x)

x: aexp

Meldet die Quadratwurzel des numerischen Ausdrucks x zurück. Wenn der Ausdruck x kleiner als Null ist, erzeugt dies eine Fehlermeldung.

Beispiele:

```
x=9
y=SQR(x)
PRINT y,SQR(4 *4)
```

Es erscheinen die Zahlen 3 und 4 auf dem Bildschirm.

```
PRINT SQR(SQR(16))
PRINT SQR(-2)
```

Es erscheint die Zahl 2 und dann eine Fehlermeldung.

EXP(x)
LOG(x)
LOG10(x)

x: aexp

EXP berechnet die x-te Potenz zur Basis der Euler'schen Zahl (e= 2.1782818284...) und LOG bildet die Umkehrfunktion dazu, bestimmt also den Logarithmus von x zur Basis e (natürlicher Logarithmus). Analog dazu gibt LOG10 den Logarithmus von x zur Basis 10 zurück (dekadischer Logarithmus).

Der numerische Ausdruck x muß bei den Logarithmusfunktionen größer als Null sein, sonst erscheint eine Fehlermeldung.

Beispiel:

```
x=2
y=EXP(2)
PRINT y,LOG10(2 *5),LOG(x)
```

Auf dem Bildschirm erscheint 7.389056098931, 1 und 0.6931471805599.

DEC i
INC i

i: avar

Die Befehle DEC und INC dienen der Veränderung eines Wertes um eins. DEC vermindert (dekrementiert) den Wert der numerischen Variable i um eins, während INC den Wert von i um eins erhöht (inkrementiert).

Diese Befehle arbeiten auch mit Fließkomma-Variablen, sind jedoch mit Integer-Variablen erheblich schneller. Wichtig ist, daß INC und DEC bei Integervariablen (%,&,|) keine Überlaufprüfung durchführen.

DEC x	entspricht	x=x–1
INC x	entspricht	x=x+1

Beispiele:

```
x%=4
y%=7
DEC x%
INC y%
PRINT x%,y%
```

Auf dem Bildschirm erscheinen die Zahlen 3 und 8.

```
a|=255
INC a|
INC a|
PRINT a|
```

Ergibt 1!, da keine Fehlerprüfung.

PRED(i)
SUCC(i)

i: iexp

PRED gibt die nächst kleinere und SUCC die nächst größere Zahl zurück. PRED und SUCC liefern also den Vorgänger (predecessor) bzw. den Nachfolger (successor) eines numerischen Ausdrucks. Beide Funktionen arbeiten mit Integerarithmetik, Nachkommastellen werden also ignoriert.

Diese beiden Funktionen stehen auch für String-Ausdrücke zur Verfügung (siehe auch im Abschnitt über Zeichenkettenverwaltung).

Beispiel:

```
i%=6
j%=PRED(i%)
PRINT j%,SUCC(2),PRED(3 *i%)
```

Auf dem Bildschirm erscheinen die Zahlen 5, 3 und 17.

ABS(x)
SGN(x)

x: aexp

Die Funktion ABS liefert den Absolutbetrag eines numerischen Ausdrucks. Dies führt zu folgenden Rückgabewerten:

x ist	ABS(x) ist dann
Negativ	–x
Gleich 0	0
Positiv	x

Mit der Funktion SGN kann man ermitteln, welches Vorzeichen ein numerischer Ausdruck hat. Es gilt:

x ist	SGN(x) ist dann
Negativ	–1
Gleich 0	0
Positiv	1

Beispiel:

```
x=-2
y=ABS(x)
PRINT SGN(x),ABS(5-3),SGN(ABS(x *3))   !ABS(x *3) ist gleich 6
```

Auf dem Bildschirm erscheinen die Zahlen –1, 2 und 1.

INT(x)
TRUNC(x)
FIX(x)
FRAC(x)

x: aexp

Diese Funktionen liefern die Vor- und Nachkommastellen von numerischen Ausdrücken. INT, TRUNC (und die mit TRUNC identische Funktion FIX) liefern eine ganze Zahl zurück. Die Funktion TRUNC schneidet die Nachkommastellen von x ab. INT liefert die größte ganze Zahl zurück, die kleiner oder gleich x ist. Der Unterschied zwischen TRUNC (FIX) und INT ist bei positiven x-Werten nicht zu erkennen. Bei einem negativen, nachkommabehafteten x entsteht jedoch ein Unterschied. So würde TRUNC bei einem x von –1.2 die Nachkommastelle entfernen und erhält den Wert –1. INT dagegen sucht die nächste ganze Zahl, die kleiner als –1.2 ist und findet dadurch –2 als Ergebnis.

FRAC liefert die Nachkommastellen von x, entfernt also die Vorkommastellen. Der durch FRAC erhaltene Wert hat dasselbe Vorzeichen wie x. FRAC ist komplementär zu TRUNC und nicht zu INT. Es gilt:

$$x = \mathrm{TRUNC}(x) + \mathrm{FRAC}(x)$$

Ersetzt man TRUNC durch INT, so stimmt die Gleichung nicht mehr im negativen Zahlenbereich (z.B. für y gleich –1.2).

Beispiel:

```
x=–1.4
y=TRUNC(1.3)
PRINT  y,INT(x),FIX(3 *x),FRAC(x–3)
```

Auf dem Bildschirm erscheinen die Zahlen 1, –2, –4 und –0.4.

ROUND(x[,n])

x: aexp
n: iexp

Die Funktion ROUND(x) liefert einen gerundeten Wert. Die Variante ROUND(x,n) rundet den Ausdruck „x" auf „n" Nachkommastellen. Ist „n" gleich Null, wird wie bei ROUND(x) auf ganze Zahlen gerundet. Ist „n" negativ, wird vor dem Komma gerundet. So ergibt ROUND(155,–1) die Zahl 160 (vgl. auch CINT()= Runden mit Integer-Ergebnis).

Beispiele:

```
y=ROUND(-1.2)
PRINT y,ROUND(1.7)
```

Gibt –1 und 2 auf dem Bildschirm aus.

```
FOR i%=-5 TO 5
  PRINT i%,ROUND(PI *100,i%)
NEXT i%
```

Gibt die Laufvariable und den zugehörigen, formatierten Ausdruck auf dem Bildschirm aus. Dazu wird die Kreiszahl Pi mit 100 multipliziert und auf i% Stellen gerundet ausgegeben. Ist i% negativ, wird vor dem Komma gerundet.

ODD(x)
EVEN(x)

x: aexp

Diese beiden Funktionen prüfen, ob der numerische Ausdruck x gerade oder ungerade ist. ODD liefert den Wert –1 (TRUE), wenn x ungerade ist und 0 (FALSE), wenn x gerade ist. EVEN ergibt –1 für ein gerades x und 0 für ein ungerades x. 0 wird wie eine gerade Zahl behandelt.

x ist	ODD(x) ist dann	EVEN(x) ist dann
Gerade	0 (FALSE)	–1 (TRUE)
Ungerade	–1 (TRUE)	0 (FALSE)
Gleich 0	0 (FALSE)	–1 (TRUE)

Beispiel:

```
x=2
PRINT ODD(x),EVEN(–2),ODD(3 *5), EVEN(–3 *x)
```

Es erscheinen die Zahlen 0, –1, –1 und 0 auf dem Bildschirm.

MIN(x [,y,z,...])
MIN(x$ [,y$,z$,...])
MAX(x [,y,z,...])
MAX(x$ [,y$,z$,...])

x, y, z: aexp
x$, y$, z$: sexp

Die Funktion MAX liefert den größten der numerischen Ausdrücke x,y,z,..., die in der Parameterliste dieser Funktion aufgeführt sind. Analog dazu bestimmt MIN den kleinsten Ausdruck. Außerdem können die Funktionen MIN und MAX ebenso auf Zeichenkettenausdrücke angewandt werden.

Beispiel:

```
x=3
y=MAX(3,5,5-4)
PRINT MIN(x,y),MAX(-1,x *2)
```

Auf dem Bildschirm erscheinen die Zahlen 3 und 6.

RND [(x)]
RANDOM(x)
RAND(y)
RANDOMIZE y

x: aexp
y: iexp

Diese Gruppe von Kommandos dient der Erzeugung von Zufallszahlen. RND erzeugt eine Zufallszahl zwischen 0 und 1 (inklusive 0, exklusive 1). Der optionale Parameter x hat keine Bedeutung.

RANDOM erzeugt eine ganzzahlige Zufallszahl zwischen 0 und x (inklusive 0, exklusive x). Der numerische Ausdruck x darf nicht ganzzahlig sein, wenn nicht alle Zahlen mit gleicher Wahrscheinlichkeit erscheinen sollen.

RAND erzeugt eine zufällige 16-Bit-Integerzahl im Bereich von 0 bis y–1. Es werden 16-Bit von y ausgewertet.

Der Befehl RANDOMIZE initialisiert den Zufallszahlengenerator mit dem Wert „y". So erhält man, wenn man den Zufallszahlengenerator mehrmals mit dem selben „y" initialisiert, danach immer dieselbe Folge von Zufallszahlen. Zu Beginn jedes Programmlaufs wird der Zufallszahlengenerator mit einer „zufällig" ausgewählten Zahl initialisiert. Ohne die Verwendung von RANDOMIZE erhält man also nach jedem Programmstart andere Zufallszahlen mit RND, RANDOM oder RAND.

Zur Initialisierung des Zufallszahlengenerators kann man RANDOMIZE auch ohne Parameter oder RANDOMIZE 0 benutzen.

Beispiele:

```
x=RND
PRINT x,RND(2)
```

Zwei Zufallszahlen zwischen 0 und 1 erscheinen auf dem Bildschirm.

```
x=RANDOM(2)
y=RAND(4)
PRINT x,y,RAND(x),RANDOM(3*x)
```

Vier ganzzahlige Zufallszahlen erscheinen auf dem Bildschirm.

```
RANDOMIZE 3
x=RND
RANDOMIZE 3
PRINT x,RND
```

Es erscheint zweimal die gleiche „Zufallszahl" auf dem Bildschirm.

BCLR(x,y)
BSET(x,y)
BCHG(x,y)
BTST(x,y)

x, y: iexp

Diese Befehle erlauben das Löschen, Setzen, Negieren und Testen eines Bits. Dabei beginnt die Zählung der Bits bei Null. Die Bit-Nummer liegt im Bereich 0 bis 31 und wird Prozessor-intern durch AND 31 maskiert.

Dabei gilt folgende Zählweise: 0 ist das niederstwertige Bit; bei 4-Byte-Integerwerten ist 31 das höchstwertige Bit und zugleich das Vorzeichen-Bit (ist das Vorzeichen-Bit gesetzt, wird die negative Zahl im 2er-Komplement dargestellt, sonst ist es eine positive Zahl).

Die Funktion BCLR setzt das y-te Bit des numerischen Ausdrucks x gleich null. BSET setzt analog dazu Bit-Nummer y von x gleich 1. BCHG setzt Bit y von x gleich 1, wenn es vorher 0 war und gleich 0, wenn es vorher 1 war. Die Funktion BTST ergibt –1 (TRUE), wenn Bit y von x gleich 1 ist und 0 (FALSE), wenn dieses Bit gleich 0 ist.

Beispiele:

```
x=BSET(0,3)
PRINT x,BSET(0,5)
```

Es erscheinen die Zahlen 8 (2^3) und 32 (2^5) auf dem Bildschirm.

```
REPEAT
  t|=Inp(2)
  PRINT CHR$(t|),CHR$(BCLR(t|,5))
UNTIL CHR$(t|)="x"
```

Beim Drücken einer Buchstabentaste erscheint der Buchstabe und der dazugehörende groß geschriebene Buchstabe (bei Kleinbuchstaben ist Bit 5 immer gesetzt, das Löschen dieses Bits erzwingt die Umwandlung in einen Großbuchstaben).

```
s$="Testwort"
FOR i%=1 TO LEN(s$)
  PRINT CHR$(BCHG(ASC(MID$(s$,i%)),5));
NEXT i%
```

Gibt „tESTWORT“ auf dem Bildschirm aus, verwandelt also jeden Klein- in einen Großbuchstaben und umgekehrt. Dies gilt jedoch nicht für Umlaute.

SHL(x,y) **SHL&(x,y)** **SHL|(x,y)**
SHR(x,y) **SHR&(x,y)** **SHR|(x,y)**

x, y: iexp

Diese Befehle verschieben (SHift) den Inhalt eines numerischen Ausdrucks x um y Bit. Ohne Angabe eines speziellen Variablentyps geschieht dies auf Langwortlänge (4 Bytes), bei Angabe eines „&“ auf Wortlänge (2 Bytes) und bei Angabe eines „|“ auf Bytelänge. Der dritte Buchstabe des Funktionsnamens gibt die Richtung der Verschiebung oder Rotation an. Dabei steht „L“ für links (Left) und „R“ für rechts (Right).

Bei Word-Funktionen(&) wird das Bit 15 anschließend in die Bits 16 bis 31 kopiert, und bei Byte-Funktionen(|) werden die Bits 8 bis 31 gelöscht.

Die folgenden Zahlenbeispiele zeigen die Wirkung der Shift-Befehle.

x%	**SHL|(x,1)**	**BIN$(x%,16)**	**BIN$(SHL|(x%,1),16)**
18	36	00000000 00010010	00000000 00100100
642	4	00000010 10000010	00000000 00000100

x%	**SHL&(x%,1)**	**BIN$(x%,16)**	**BIN$(SHL&(x%,1),16)**
18	36	00000000 00010010	00000000 00100100
130	4	00000000 10000010	00000001 00000100

x%	**SHR&(x%,2)**	**BIN$(x%,16)**	**BIN$(SHR&(x%,2),16)**
24	6	00000000 00011000	00000000 00000110
4162	1040	00010000 01000010	00000100 00010000

(Die Bits sind nur aus Gründen der Übersichtlichkeit in Achter-Gruppen zusammengefaßt worden, BIN$ gruppiert nicht.)

ROL(x,y) **ROL&(x,y)** **ROL|(x,y)**
ROR(x,y) **ROR&(x,y)** **ROR|(x,y)**

x, y: iexp

Für die Operandenlänge und dir Richtung der Verschiebung gilt das gleiche, wie für die Shift-Befehle. Es sei z.B. nur das höchste Bit eines Bytes gesetzt. Dieses Byte wird nun um ein Bit nach links rotiert (ROL|(128,1)). Das links herausgeschobene Bit wird nun rechts wieder hereingeschoben, so daß das erste Bit des Funktionsergebnisses gesetzt ist. Bei SHL|(128,1) wäre das herausgeschobene gesetzte Bit gleich Null geworden.

Weitere Zahlenbeispiele sind:

x|	ROL|(x,1)	BIN$(x|,8)	BIN$(ROL|(x|,1),8)
6	12	00000110	00001100
130	5	10000010	00000101

x|	ROR|(x|,3)	BIN$(x|,8)	BIN$(ROR|(x|,3),8)
66	144	01000010	00000000
2	64	00000010	01000000

Beispiel:

```
x|=128+1                !setzt Bit 7 und 0
y%=ROR|(x|,1)           !y wird gleich 192
PRINT SHL(y%,4),y%*2^4
PRINT SHL(ROR|(128+1,1),4)
```

Es erscheint dreimal die Zahl 3072 auf dem Bildschirm. Dem Ausdruck a*2^b entspricht die Funktion SHL(a,b), sofern kein Bit über den vier Byte langen Geltungsbereich geschoben wird. Die Formulierung, die die Bit-Funktion benutzt, ist allerdings deutlich schneller.

AND(x,y)
OR(x,y)
XOR(x,y)

x, y: iexp

Diese Funktionen sind logische Verknüpfungen zweier numerischer Ausdrücke. Im Funktionsergebnis von AND sind nur Bits gesetzt, die sowohl in x als auch in y gesetzt sind. Das Funktionsergebnis von OR enthält gesetzte Bits an den Stellen, an denen Bits in x oder y oder in beiden Ausdrücken gesetzt sind. XOR setzt nur Bits, die in x oder y, aber nicht in x und y gleichzeitig gesetzt sind. Anders formuliert: XOR setzt Bits, die in x und y verschiedene Werte haben.

Beispiel:

```
x=3
y=2
z=AND(x,y)   !z wird gleich 2
PRINT OR(2,7),XOR(x,1+4+8)
```

Auf dem Bildschirm erscheinen die Zahlen 7 und 14.

IMP(x,y)
EQV(x,y)

x, y: iexp

IMP weist Bits nur dann eine Null zu, wenn das entsprechende Bit in x gesetzt und in y nicht gesetzt ist, ansonsten wird das Bit im Funktionsergebnis gesetzt. EQV setzt ein Bit des Funktionsergebnisses, wenn die entsprechenden Bits in x und y dieselben Werte haben. Betrachten Sie zur Erläuterung dieser Kommandos auch die Wahrheitstabellen im Abschnitt über logische Operatoren.

Beispiel:

```
PRINT BIN$(15,4),15
PRINT BIN$(6,4),6
PRINT BIN$(IMP(15,6),4),"IMP(15,6)"
PRINT BIN$(EQV(15,6),4),"EQV(15,6)"
```

Auf dem Bildschirm erscheint:

```
1100  15
0101  6
0111  IMP(15,6)
0110  EQV(15,6)
```

SWAP(x)

x: iexp

Die Funktion SWAP faßt den numerischen Ausdruck x als Langwort (4 Byte) auf und vertauscht dessen oberes und unteres Wort (jeweils 2 Byte). Diese Funktion hat nichts mit dem gleichnamigen GFA-BASIC-Befehl zu tun und wird für verschiedene Zwecke gebraucht (z.B. um einen Langwortparameter in zwei Worten an eine Betriebssystemroutine zu übergeben oder um Pointer mit vertauschter Wortreihenfolge zu bearbeiten).

Beispiel:

```
x=1044480
PRINT BIN$(x,32)
y=SWAP(x)
PRINT BIN$(y,32)
```

Auf dem Bildschirm erscheint:

```
00000000000011111111000000000000
11110000000000000000000000001111
```

BYTE(x)
CARD(x)
WORD(x)

x: iexp

BYTE gibt die unteren 8 Bits des numerischen Ausdrucks x zurück. Entsprechend liest CARD die unteren 16 Bits von x. WORD erweitert ein Wort auf Langwortlänge (d.h. Bit 15 wird in Bit 16 bis Bit 31 kopiert).

Beispiel:

```
PRINT BYTE(1+254),BYTE(1+255)
PRINT HEX$(CARD(&H1234ABCD))
```

Schreibt 255, 0 und ABCD auf den Bildschirm.

LEFT$(a$ [,x])
RIGHT$(a$ [,x])

a$: sexp
x: iexp

LEFT$ liefert die ersten x-Zeichen der Zeichenkette a$. Ist x größer als die Anzahl der Zeichen in a$, so wird a$ komplett übergeben. Ohne die Angabe von x wird nur das erste Zeichen des Strings a$ gemeldet. RIGHT$ ergibt analog dazu die letzten x-Zeichen von a$. Ohne Angabe von x wird das letzte Zeichen von a$ bestimmt.

Beispiele:

```
a$="Handbuch für GFA–BASIC"
b$=LEFT$("GFA–Systemtechnik",4)
PRINT b$;RIGHT$(a$,5)
```

Es wird das Wort „GFA–BASIC“ auf den Bildschirm geschrieben.

```
a$="Oberhausen"
b$=LEFT$(a$)+RIGHT$("Technik")
PRINT b$
```

Es erscheint „Ok“ auf dem Bildschirm.

MID$(a$,x [,y]) (als Funktion)

a$: sexp
x, y: iexp

Die Funktion MID$ gibt y-Zeichen ab der Position x des Strings a$ zurück. Ist x größer als die Länge von a$, so wird eine leere Zeichenkette zurückgegeben. Läßt man y weg, so wird der gesamte Teil-String ab dem x-ten Zeichen ermittelt.

Beispiel:

```
a$="Handbuch für GFA–BASIC"
b$=MID$(a$,14,9)+MID$("Version 3.0",8)
PRINT b$
```

Es erscheint der Text „GFA-BASIC 3.0“ auf dem Bildschirm.

MID$(a$,x [,y]) (als Befehl)

a$: svar
b$: sexp
x, y: iexp

MID$ als Befehl ermöglicht das Einsetzen eines String-Ausdrucks in die Mitte einer String-Variable. So würde bei MID$(a$,x,y)=b$ in die Variable a$ ab dem x-ten Zeichen der String-Ausdruck b$ eingesetzt. Der optionale Parameter y legt fest, wie viele Zeichen maximal von b$ in a$ eingesetzt werden. Wird y weggelassen, so werden so viele Zeichen wie möglich in a$ übernommen. Durch MID$ wird die Länge von a$ nicht verändert, sondern die aus b$ kommenden Zeichen überschreiben Teile von a$. Ist a$ zu kurz, um b$ ganz aufzunehmen, so wird das Einsetzen abgebrochen.

Beispiel:

```
a$="GF    ASIC"
MID$(a$,3)="A-B"
b$="Testwort"
MID$(b$,5,2)="wer war das ?"
PRINT a$,b$
```

Es erscheinen die Worte „GFA-BASIC“ und „Testwert“.

INSTR(a$,b$)
INSTR(a$,b$, [x])
INSTR([x],a$,b$)

a$, b$: sexp
x: iexp

Die Funktion INSTR sucht in der Zeichenkette a$ nach dem String b$. Ist x angegeben, so wird in a$ ab dem x-ten Zeichen mit der Suche begonnen, ansonsten beginnt die Suche beim ersten Zeichen. Es wird die Position zurückgemeldet, ab der b$ in a$ ent halten ist. Ist b$ nicht gefunden worden, so ist INSTR gleich Null. Wenn a$ und b$ Leer-Strings sind, ist INSTR gleich Eins.

Beispiel:

```
a$="GFA–Systemtechnik"
x=INSTR(a$,"System")
PRINT x,INSTR("GFA–BASIC","BASIC",6)
```

Es erscheinen die Zahlen 5 und 0 auf dem Bildschirm.

RINSTR(a$,b$)
RINSTR(a$,b$, [x])
RINSTR([x],a$,b$)

a$, b$: sexp
x: iexp

RINSTR() durchsucht wie INSTR einen String, beginnt allerdings die Suche am Ende der Zeichenkette oder an der angegebenen Position und sucht dann nach vorne.

Beispiel:

```
PRINT RINSTR("DF0:ORDNER/ TEST.GFA","/")
```

Sucht nach dem letzten Schrägstrich „ / " im Pfadnamen und gibt die Position des gesuchten Zeichens (in diesem Fall 11) aus.

STRING$(x,a$)
STRING$(x,code)
SPACE$(x)
SPC(x)

x, code: iexp
a$: sexp

Die Funktion STRING$ erzeugt einen String, der den String-Ausdruck „a$" oder den ASCII-Wert „code" insgesamt x-mal (zwischen 0 und 32767) enthält. Bei Angabe von „code" wird CHR$(code) verwendet.

SPACE$ erzeugt einen String, der x Leerzeichen enthält. SPC gibt in einem PRINT-Befehl x Leerzeichen aus. SPC kann im Unterschied zu SPACE$ keinen String erzeugen, den man z.B. über ein Gleichheitszeichen einer Variablen zuweisen könnte.

Beispiel:

```
a$="GFA  "
b$=SPACE$(5)
PRINT  b$;STRING$(3,a$);SPC(4);STRING$(5,"<")
```

Auf dem Bildschirm erscheint „ GFA GFA GFA <<<<<".

PRED(a$)
SUCC(a$)

a$: sexp

PRED liefert das Zeichen mit dem um eins verringerten ASCII-Wert des zu übergebenden String-Ausdrucks. Dabei wird jeweils nur das erste Zeichen einer Zeichenkette ausgewertet. Analog liefert SUCC das um eins erhöhte Zeichen. Dabei entspricht PRED(a$) dem Ausdruck CHR$(PRED(ASC(a$))) und SUCC(a$) entspricht CHR$(SUCC(ASC(a$))).

Diese beiden Funktionen stehen auch für Integerausdrücke zur Verfügung.

Beispiel:

```
zeichen$="B"
vorgaenger$=PRED(zeichen$)
nachfolger$=SUCC(zeichen$)
PRINT  vorgaenger$,zeichen$,nachfolger$
```

Gibt die Buchstaben A, B und C auf dem Bildschirm aus.

LEN(a$)
TRIM$(a$)

a$: sexp

LEN ermittelt, wie viele Zeichen in a$ enthalten sind und meldet diese Zahl zurück.

TRIM$ entfernt Leerzeichen am linken und rechten Rand eines Strings.

Beispiele:

```
a$="Test"
x=LEN(a$)+1
PRINT x,LEN("Wort")
```

Es erscheinen die Zahlen 5 und 4 auf dem Bildschirm.

```
b$="   test   "
PRINT LEN(b$)
PRINT TRIM$(b$)
PRINT LEN(TRIM$(b$))
```

Ausgegeben wird 10, der String „test“ ohne Leerzeichen und 4.

UPPER$(a$)

a$: sexp

Wandelt alle kleingeschriebenen Buchstaben einer Zeichenkette in Großbuchstaben um. Dies funktioniert auch mit Umlauten.

Beispiel:

```
a$="Test"
b$=UPPER$(a$)+UPPER$("Wo"+"rt")
PRINT b$;UPPER$(" Gfa-Basic 3.0")
```

Auf dem Bildschirm erscheint „TESTWORT GFA-BASIC 3.0“.

LSET a$=b$
RSET a$=b$

a$: svar
b$: sexp

LSET und RSET setzen den String-Ausdruck b$ links- (LSET) bzw. rechtsbündig in die String-Variable a$ ein und füllen gegebenenfalls mit Leerzeichen auf.

Beispiele:

```
a$="      "
FOR i%=1 TO 128              ! gibt linksbündig formatierte
  LSET a$=STR$(i%)
  PRINT a$;
NEXT i%
PRINT
FOR i%=1 TO 128
  RSET a$=STR$(i%)           ! und rechtsbündig formatierte
  PRINT a$;                  ! Zahlenkolonnen aus
NEXT i%
```

Gibt linksbündig und rechtsbündig formatierte Zahlenkolonnen aus. Statt RSET a$=STR$(i%) kann auch a$=STR$(i%,5,0) verwendet werden.

5 – Ein- und Ausgabe

INKEY$

INKEY$ liest ein Zeichen von der Tastatur. Diese Funktion ist allerdings nicht in der Lage, die Tastaturumschalttasten abzufragen (Shift, Alternate, Control, CapsLock). Die Funktion meldet das ASCII-Zeichen der gedrückten Taste zurück. Bei Sondertasten, wie den Funktionstasten, wird die zugehörige CSI-Sequenz zurückgegeben. D.h., CHR$(155) gefolgt von einer Anzahl weiterer Zeichen.

Folgendes Beispiel gestattet die Ermittlung der mit INKEY$ zu erhaltenden Werte.

Beispiel:

```
DO
  t$=INKEY$
  IF t$<>""
    IF LEN(t$)=1
      PRINT "ASCII–Code: ";ASC(t$),"ASCII–Zeichen: ";t$
    ELSE
      FOR I%=1 TO LEN(A$)
        PRINT ASC(MID$(A$,F%))'
      NEXT I%
      PRINT
    ENDIF
  ENDIF
LOOP
```

Zeigt bei jeder gedrückten Taste ihren ASCII-Code oder die zugehörige CSI-Sequenz an.

INPUT ["text",] x [,y,...]
INPUT ["text";] x [,y,...]

x, y: avar oder svar

Der Befehl INPUT kann in mehreren Variationen verwendet werden. Er dient der Eingabe von Variablen oder Variablenlisten mit oder ohne vorangestellte Erläuterungen („text"). INPUT wird immer auf einem Fenster ausgeführt. Ist im Programm noch kein Fenster geöffnet worden, so wird automatisch das Fenster 0 mit der Standardgröße geöffnet.

Der Befehl bezieht sich immer auf die letzte Cursor-Position des aktuellen Ausgabefensers. Mit Hilfe von PRINT AT, gefolgt von einem Semikolon oder LOCATE, VTAB, HTAB, kann der Cursor plaziert werden, um die Position der Eingabe festzulegen.

Wenn dem Befehlswort INPUT ein Text folgt, so kann dieser von den nachfolgenden Variablen durch ein Komma oder ein Semikolon getrennt werden. Wird ein Semikolon verwendet, so werden ein Fragezeichen und ein Leerzeichen an die Beschreibung angehängt und der Cursor dahinter plaziert. Verwendet man ein Komma, wird der Cursor für die Eingabe direkt hinter dem letzten Zeichen der Beschreibung plaziert.

Wenn kein Text hinter INPUT steht, so erscheint in jedem Fall ein Fragezeichen, gefolgt von einem Leerzeichen, hinter dem der Cursor steht.

Wird nur eine einzige Variable abgefragt, so muß deren Eingabe durch das Drücken der Return- oder Enter-Taste bestätigt werden.

Wenn mehrere Variablen mit einem INPUT-Befehl erfragt werden, so kann jede einzelne Variable durch Drücken von Return oder Enter bestätigt werden. Die Variablen können aber auch durch Kommas getrennt und mit einem einzigen Druck der Return- oder Enter-Taste eingegeben werden.

Wenn man dennoch Kommas in einem String per INPUT eingeben will, muß man den Befehl LINE INPUT verwenden.

Wurde eine numerische Variable erwartet, aber ein Text eingegeben, so wird ein Beep ausgeführt, und die Eingabe muß erneut begonnen werden. Bis zum Drücken der Return- oder Enter-Taste können mit Hilfe von Backspace- und Delete-Taste Zeichen innerhalb der Eingabe gelöscht werden; auch die linke und rechte Cursor-Taste sind aktiv.

Beispiel:

```
OPENW 0
INPUT a$
INPUT "",b$
INPUT "Zwei Zahlen bitte: ";x,y
PRINT a$,b$,x,y
```

Liest zwei Strings und zwei numerische Variablen ein. Der erste INPUT-Befehl meldet sich mit „?“, der zweite ohne Text und der dritte mit der Aufforderung „Zwei Zahlen bitte: ?“

LINE INPUT ["text",] a$ [,b$...]
LINE INPUT ["text";] a$ [,b$...]

a$, b$: svar

LINE INPUT ist eine Variante des INPUT-Befehls. Im Unterschied zu INPUT ist es hier möglich, Kommas in einer String-Variablen einzugeben. Vorangehende Beschreibungen, Eingabe von Variablen oder einer Variablenliste, die Korrektur der Eingabe bis zum Drücken der Return- oder Enter-Taste und weitere Variationsmöglichkeiten werden wie bei INPUT gehandhabt. Es ist allerdings nur für String-Variablen möglich.

Beispiel:

```
OPENW 0
LINE INPUT a$
INPUT b$
PRINT a$,b$
```

Geben Sie bitte zweimal den Text „Kom,ma“ ein. Es erscheint danach „Kom,ma“ und „Kom“ auf dem Monitor.

Vgl. auch LINE INPUT # (Kapitel 6).

FORM INPUT n,a$
FORM INPUT n AS a$

n: iexp
a$: svar

FORM INPUT und FORM INPUT AS dienen beide der Eingabe von String-Variablen. Dabei gibt n an, wie viele Zeichen maximal im String a$ sein dürfen (zwischen 1 und 255).

FORM INPUT AS gibt zusätzlich den aktuellen Wert von a$ aus, der dann editiert werden kann. Die Editiermöglichkeiten sind bei beiden Befehlen dieselben wie beim INPUT-Befehl.

Beispiel:

```
FORM INPUT 10,a$
b$="test"
 FORM INPUT 5 AS b$
PRINT a$,b$
```

Zwei Strings werden abgefragt. Bei der Abfrage des zweiten Strings wird das Wort „test“ als voreingestellter Wert von b$ zur Editierung freigegeben.

PRINT
PRINT ausdruck
PRINT AT(spalte,zeile);ausdruck
WRITE ausdruck
LOCATE zeile,spalte

ausdruck: beliebige sexp oder aexp oder deren Kombination
spalte, zeile: iexp

Der Befehl PRINT und seine Varianten erzeugen eine Bildschirmausgabe. Beim Amiga muß dazu unbedingt ein Fenster geöffnet sein. Wenn dies im Programm bereits geschehen ist, erfolgt die Ausgabe auf das Fenster, auf das zuletzt ein OPENW-Befehl ausgeführt wurde. Ansonsten wird ein Standard-Fenster geöffnet. Der Befehl PRINT ohne folgende Parameter bewirkt einen Zeilenvorschub. Befindet sich der Cursor bereits in der letzten Zeile, wird der gesamte Fensterinhalt um eine Zeile nach oben verschoben. PRINT, von einem Ausdruck gefolgt, bewirkt die Ausgabe dieses Ausdrucks an der aktuellen Cursor-Position. Zeichenketten müssen in Anführungszeichen eingeschlossen werden. Besteht der auszugebende Ausdruck aus mehreren Elementen (Konstanten, Variablen oder Ausdrücken), so können die einzelnen Teile durch Semikolon, Komma oder Apostroph getrennt werden.

Die Verwendung des Kommas bewirkt die Plazierung des Cursors hinter die nächste, durch 16 teilbare, Spaltenposition. Ist die letzte Spalte erreicht, wird der Cursor in Spalte 17 der nächsten Zeile bewegt. Das Semikolon bewirkt die Ausgabe der betreffenden Elemente ohne Zwischenraum. Verwendet man das Apostroph, wird ein Leerzeichen zwischen den entsprechenden Elementen eingefügt. Das Apostroph ist auf der deutschen Tastatur das ALT-"A". Es läßt sich aber auch durch die Akzentzeichen erreichen – z.B. die Taste neben der Zahl 1 – die automatisch umgewandelt werden.

PRINT AT ermöglicht die Positionierung des auszugebenden Ausdrucks ab einer bestimmten Spalte und Zeil. Dabei stehen, je nach Auflösung, bis zu 80 Spalten und 57 Zeilen zur Verfügung. Wenn der Ausgabeausdruck nicht mit einem Semikolon beendet wird, so wird der Cursor an den Anfang der nächsten Zeile plaziert. Falls er schon in der letzten Zeile stand, so wird der Fensterihalt um eine Zeile nach oben geschoben.

Im Gegensatz zu PRINT AT setzt LOCATE lediglich den Cursor an die angegebene Zeilen- und Spaltenposition. Es ist nicht möglich, Ausdrücke mit LOCATE auszugeben (Vgl. VTAB/HTAB).

Der WRITE-Befehl dient der Datenspeicherung in sequentiellen Dateien zum Einlesen mit Input. Nach dem WRITE-Befehl stehen numerische und String-Ausdrücke, die durch Kommata voneinander getrennt sind.

Bei der Ausgabe werden die Ausdrücke auch durch Kommata voneinander getrennt; String-Ausdrücke werden in Anführungszeichen eingeschlossen.

Anmerkung: Dieses Format eignet sich bei Ausgabe auf Diskette besonders für das Wiedereinlesen mit INPUT. Hinter dem letzten Ausdruck eines WRITE-Befehls kann auch ein Semikolon stehen. Wodurch die abschließende Ausgabe von CR/LF unterdrückt wird.

Beispiele:

```
OPENW 0
a$="GFA–Systemtechnik"
PRINT Left$(a$,4)+"BASIC"'1+2;
PRINT ".0","GFA–";UPPER$(MID$(a$,5))
```

Auf dem Bildschirm erscheint der Text „GFA–BASIC 3.0“ und „GFA–SYSTEMTECHNIK“.

```
PRINT AT(4,8);"in der vierten Spalte der achten Zeile"
```

Schreibt einen String an die Position 4, 8.

```
LOCATE 8,4
PRINT "In der achten Zeile, an der vierten Spalte"
```

Positioniert den Cursor an die vierte Spalte der achten Zeile und gibt mit dem darauffolgenden PRINT an dieser Stelle einen String aus.

```
WRITE 1+1,"Hallo",3 *4
```

Ergibt die Ausgabe 2,"Hallo",12

PRINT USING format$,ausdruck [;]
PRINT AT(zeile,spalte);USING format$,ausdruck [;]

format$:	sexp
ausdruck:	beliebig viele,durch Kommata getrennte, sexp oder aexp
spalte, zeile:	iexp

PRINT USING und seine Variation PRINT AT USING dienen der formatierten Datenausgabe im Fenster. Grundsätzlich arbeiten diese Befehle wie PRINT bzw. PRINT AT. Jedoch werden die im auszugebenden Ausdruck stehenden Daten gemäß dem Inhalt von format$ formatiert.

Zur Formatierung von numerischen Ausdrücken stehen folgende Zeichen zur Verfügung:

Platzhalter für eine Ziffer. Ist es die letzte Ziffer der Formatanweisung, so wird bei der Ausgabe aufgerundet.

. Dient zur Abtrennung der Dezimalziffern innerhalb mehrerer #-Zeichen.

, Fügt ein Komma an der jeweiligen Stelle zwischen den #-Zeichen ein und bewirkt so z.B. eine Trennung in den Tausender-Stellen.

– Das Minus-Zeichen darf nur an erster oder letzter Stelle des Format-Strings stehen. Es reserviert eine Stelle für die Ausgabe eines negativen Vorzeichens. Bei einem positiven Vorzeichen wird stattdessen ein Leerzeichen ausgegeben.

+ Analog zum Minus-Zeichen wird bei positiven Zahlen ein Plus-Zeichen vor- oder hintenangestellt. Plus- und Minus-Zeichen können nicht kombiniert werden.

* Ersatz für #, führende Nullen werden anstatt durch Leerzeichen durch Sternchen ersetzt.

$ Bewirkt die Ausgabe eines $-Zeichens vor der Zahlenangabe, wenn es unmittelbar vor dem ersten #-Symbol steht.

^ Dient der Einstellung des Exponential-Formats (E+000). Dabei bestimmen die #-Zeichen die Länge der Mantisse. Die ^-Zeichen geben die Anzahl der Stellen für den Exponenten inklusive E+ bzw. E– an. Stehen mehrere # vor dem Dezimalpunkt, so wird der Exponent so angepaßt, daß er durch diese Anzahl teilbar ist. Für negative Zahlen muß unbedingt ein Vorzeichen vorgesehen werden.

Für die Formatierung von Zeichenketten stehen folgende Zeichen zur Verfügung:

& Bewirkt die Ausgabe der gesamten Zeichenkette.

! Beschränkt die Ausgabe auf das erste Zeichen des Strings.

\ ...\ Gibt an, wie viele Zeichen (einschließlich der beiden \-Zeichen) einer Zeichenkette ausgegeben werden.

_ Der Unterstrich bewirkt die Ausgabe des folgenden Zeichens der Formatangabe. Weiterhin ist die unveränderte Ausgabe von Texteinschüben zwischen diesen Formatangaben möglich. Die dem Format-String folgenden Variablen, Variablenlisten oder Ausdrücke werden durch Kommas getrennt.

Beispiel:

```
OPENW 0
PRINT USING "#.####",PI
PRINT AT(4,4); USING "PI_._._. #.###",PI;
```

Es erscheinen die Texte „3.1416“ und „PI... 3.142“.

```
FOR i%=1 TO 14
  PRINT USING "###.##^^^^ ",2^i%;
NEXT i%
```

Ausgabe:

1.00E+00	2.00E+00	4.00E+00	8.00E+00
16.00E+00	32.00E+00	64.00E+00	128.00E+00
256.00E+00	512.00E+00	1.02E+03	2.05E+03
4.10E+03	8.19E+03	16.38E+03	

MODE n

n: iexp

Mit MODE können Kommata und Punkte ausgetauscht werden. Durch MODE kann zwischen Dezimalpunkt und Tausender-Komma sowie Dezimalkomma und Tausender-Punkt gewählt werden. Außerdem wird mit MODE das Format der Datumsdarstellung gewählt. Punkt und Komma gelten für PRINT USING und STR$(x,v,n). Die Datumseinstellung gilt für DATE$, SETTIME, DATE$= und FILES.

Der Parameter n kann Werte zwischen 0 und 3 annehmen. Dafür ergeben sich dann die Einstellungen gemäß folgender Tabelle:

Parameter n	USING	DATE$
MODE 0	, ###.##	16.05.1988
MODE 1	#,###.##	05/16/1988
MODE 2	#.###,##	16.05.1988
MODE 3	#.###,##	05/16/1988

DEFNUM n

n: iexp

DEFNUM beeinflußt die Ausgabe von Zahlen durch den PRINT-Befehl und dessen Varianten. Alle Zahlenausgaben, die hinter einem DEFNUM-Befehl folgen, werden mit n Stellen durchgeführt (Vor- und Nachkommastellen, der Punkt wird nicht mitgerechnet). Die interne Rechengenauigkeit wird dadurch nicht verändert. Die n+1'ste Stelle wird bei diesen Ausgaben zur Rundung herangezogen.

Beispiel:

```
OPENW 0
PRINT 100/3
DEFNUM 5
PRINT 100/3
```

Auf dem Bildschirm erscheinen die Zahlen 33.333333333 und 33.333.

CRSCOL
CRSLIN
POS(x)
TAB(n)
HTAB spalte
VTAB zeile

n, spalte, zeile: iexp
x: aexp

Die Befehlsgruppe CRSCOL, CRSLIN, POS und TAB dient der Abfrage und Plazierung des Cursors. CRSCOL liefert die aktuelle Spaltenposition und CRSLIN die aktuelle Zeilenposition des Cursors. POS liefert die Zahl der seit dem letzten Wagenrücklauf (Carriage Return) am Bildschirm ausgegebenen Zeichen AND 255. Der Ausdruck x wird ignoriert.

Der von POS zurückgelieferte Wert muß jedoch nicht mit der aktuellen Cursor-Position übereinstimmen. Wird z.B. eine Zeichenkette mit 120 Zeichen ausgegeben, so steht der Cursor in der Spalte 40, POS liefert jedoch den Wert 120. Insbesondere bei Ausgabe von Steuerzeichen hat POS(0) wenig mit der Cursorspalte zu tun, da nur die Zeichen gezählt werden. Dabei wird ein LF (CHR$(10)) nicht beachtet, ein CR (CHR$(13)) setzt den Zähler auf Null zurück, und ein BS (CHR$(8)) vermindert den Zähler um eins.

TAB(n) gibt soviele Leerzeichen aus, bis POS(0) den Wert von n erreicht. Ist n bereits überschritten, so erfolgt zunächst ein Zeilenvorschub (CR/LF) wie bei PRINT ohne Parameter. Wieder wird nur AND 255 verwendet.

Die Befehle HTAB und VTAB positionieren den Cursor an einer bestimmten Spalten- bzw. Zeilenposition. Bei Cursor-Positionierungen hat die obere linke Ecke des Fensters die Koordinate 1.

Beispiele:

```
PRINT AT(38,12);"Test ";
PRINT CRSCOL'CRSLIN PRINT Tab(37);"Test ";
PRINT POS(0)
```

Auf dem Bildschirm erscheint in der Bildschirmmitte „Test 43 12“ und „Test 42“.

```
PRINT AT(4,3);"Wort 1"
HTAB 4
VTAB 2
PRINT "Wort 2"
```

Gibt an der vierten Spalte der dritte Zeile den Text „Wort 1“ und an der vierten Spalte der zweiten Zeile den String „Wort 2“ aus.

DATA konst [,konst1,konst2,...]
READ var [,var1,var2,...]
RESTORE [mar]

konst, konst1, konst2: numerische- oder String-Konstante
var, var1, var2: avar oder svar
marke: vom Benutzer definierter Name

Mit Hilfe von DATA können konstante Werte platzsparend angegeben werden. Das Einlesen dieser Daten geschieht mit READ. Dabei können numerische Werte in hexadezimaler, oktaler oder binärer Form aufgeführt werden. Wenn eine Zeichenkette eingelesen werden soll, die Kommata enthält, so muß die gesamte Zeichenkette in Anführungsstrichen stehen.

Zu den Befehlen DATA und READ gehört ein sogenannter Datenzeiger. Dieser zeigt immer auf den nächsten mit READ zu lesenden DATA-Wert. Er ist zu Beginn des Programms der erste Wert hinter der ersten DATA-Zeile. Mit Hilfe des RESTORE-Befehls kann dieser Datenzeiger auf bestimmte DATA-Zeilen gerichtet werden. Zu diesem Zweck wird vor dieser DATA-Zeile eine Marke gesetzt und der Datenzeiger mit RESTORE mar gesetzt. Folgt dem RESTORE-Befehl keine Marke, wird der Datenzeiger auf den ersten im Programm stehenden DATA-Wert gerichtet.

Die Zeichenfolge „mar“ kann aus Ziffern, Buchstaben, Unterstrichen und Punkten bestehen und, im Gegensatz zu Variablennamen auch mit einer Ziffer beginnen. Die Marke selbst muß mit einem Doppelpunkt enden.

Beispiel:

```
OPENW 0
FOR i=1 TO 3
  READ a
  PRINT a'
NEXT i
'
RESTORE roemische_ziffern
READ a$,b$,c$,d$
PRINT
PRINT a$'b$'c$'d$
'
DATA 1,2,3,4
DATA a,b,c,d
'
roemische_ziffern:
DATA I,II,III,IV
```

In einer Schleife werden die numerischen Werte 1,2 und 3 eingelesen und auf dem Bildschirm angezeigt. Danach wird mittels RESTORE der Datenzeiger auf die DATA-Zeile mit den römischen Zahlzeichen gerichtet. Dies geschieht mit roemische_ziffern. Anschließend werden diese Daten den String-Variablen a$,b$,c$ und d$ zugewiesen und ausgegeben.

```
DATA 10,&A,$A,&HA,&O12,&X1010,%1010
FOR i%=1 to 7
  READ a%
  PRINT a%
NEXT i%
```

Liest siebenmal die Zahl 10 ein. Bei READ/INPUT/VAL usw. können HEX-Zahlen auch durch $ gekennzeichnet werden und BIN-Zahlen auch durch %, nicht nur durch &H oder & bzw, &X.

MOUSE mx,my,mk
MOUSEX
MOUSEY
MOUSEK

Die Befehle MOUSEX, MOUSEY und MOUSEK ermöglichen die Abfrage der aktuellen Mausposition und des Maustastenstatus. Mit Hilfe des Befehls MOUSE kann man diese Abfragen zusammenfassen. Dann liefert MOUSE in „mx“ und „my“ die aktuellen Mauskoordinaten und in mk den Maustastenstatus. Dabei kann die Variable „mk“ Werte zwischen 0 und 7 annehmen.

mk	Gedrückte Taste
0	Keine
1	Linke Maustaste
2	Rechte Maustaste
3	Linke und rechte Maustaste
4	Mittlere Maustaste

Bei weiteren Kombinationen der obigen Aufstellung enthält mk jeweils die Summe der Einzelwerte.

Die zurückgelieferten Koordinaten beziehen sich immer auf das aktuelle Ausgabefenster. Verläßt man den Fensterbereich mit dem Mauszeiger, erhält man negative Koordinaten. Werden die Befehle ausgeführt, ohne zuvor ein Fenster geöffnet zu haben, enthält mx und my den Wert 0.

Beispiel:

```
OPENW 0
REPEAT
  IF MOUSEK=1
    PLOT MOUSEX,MOUSEY
  ENDIF
UNTIL MOUSEK=2
'
REPEAT
  MOUSE mx%,my%,mk%
  IF mk%=2
    PLOT mx%,my%
  ENDIF
UNTIL mk%=1
```

In der REPEAT-Schleife werden Mausposition und Maustastenstatus abgefragt. Nur während die linke Maustaste gedrückt ist, wird ein Punkt an der Stelle mx,my gezeichnet. Die Schleife wird nach Drücken der rechten Maustaste verlassen. In der zweiten Schleife kann dann mit der rechten Maustaste gezeichnet und mit der linken das Programm verlassen werden.

LPRINT ausdruck
LPOS(x)

ausdruck: aexp oder sexp oder deren Kombinationen
x: avar, Scheinargument

LPRINT ausdruck ermöglicht die Ausgabe von Daten an den Drucker. Dabei ist LPRINT mit dem PRINT-Befehl identisch. Grundsätzlich können alle PRINT-Befehlsvariationen verwendet werden. Es ist aber nicht möglich, mit PRINT AT(x,y) den Druckkopf zu positionieren, aber LPRINT USING ist möglich.

LPOS(x) liefert die Anzahl der seit dem letzten CR an den Drucker ausgegebenen Zeichen (vgl. POS(x)).

Beispiele:

```
LPRINT
LPRINT "Test"
PRINT LPOS(x)
```

Führt auf dem Drucker, wenn ein solcher angeschlossen und eingeschaltet ist, einen Zeilenvorschub aus, schreibt das Wort „Test“ auf den Drucker und zeigt die aktuelle Position des Druckkopfes auf dem Bildschirm an.

HARDCOPY
HARDCOPY [adr,comap,modes,srcx,srcy,srcb,srch,destb,desth, spec]
HARDCOPY spec

adr,comap,modes,srcx,srcy,srcb,srch,destb,desth,spec: iexp

Der Befehl HARDCOPY ohne Parameter erzeugt einen Grafikausdruck des aktuellen Screens auf einem angeschlossenen Drucker. Dabei werden die Druckereinstellungen aus den System-Preferences verwendet. Ist weder ein Fenster geöffnet noch ein Rastport gesetzt, wird eine Fehlermeldung ausgegeben. Die optionalen Parameter können angegeben werden, um den Ausdruck zu verändern.

adr: Adresse des Rastports, von der die Hardcoy gemacht werden soll.

comap: Adresse der Colormap, in der die Farben des Bildes enthalten sind.

modes: Die Auflösungsmodi, wie sie in der ViewPort-Struktur angegeben werden.

srcx: Offset zum linken Rand des Rastports

srcy: Offset zum rechten Rand des Rastports.

srcb: Die Breite des auszudruckenden Bereichs.

srch: Die Höhe des auszudruckenden Bereichs.

destb: Anzahl der zu druckenden Spalten.

desth: Anzahl der zu druckenden Zeilen.

spec: Gibt an, wie die Parameter „destb“ und „desth“ ausgewertet werden. Wenn Bit 0 gesetzt ist, so ist die Angabe von „destb“ in tausendstel Zoll, Bit 1 entsprechend für „desth“. Bit 2 gesestzt bedeutet Maximalbreite, Bit 3 Maximalhöhe. Bit 4 steht dafür, daß „destb“ ein Teiler der Maximalbreite ist, Bit 5 entsprechend für die Höhe. Bit 6 (64) bedeutet, daß das Druckbild zentriert ausgegebenen wird. Bit 7 steht dafür, daß das Seitenverhältnis auf dem Papier dem auf dem Bildschirm entspricht.

In den Bits 8 bis 10 kann die Qualität des Ausdrucks bestimmt werden, z.B. &h100 für die geringste Auflösung des Druckers und &h700 für die höchste. Ein gesetztes Bit 11 unterdrückt den Seitenvorschub am Ende der Ausgabe. Default-mäßig, das heißt ohne Parameter, wird &h84 verwendet, also volle Breite bei Beibehaltung des Seitenverhältnisses.

Beispiel:

```
OPENW 2
GRAPHMODE 1
COLOR 1,2,1
DEFFILL 1,3,12
PBOX 10,10,100,100
HARDCOPY
```

Druckt den Workbench-Screen mit einem BASIC-Fenster aus.

SOUND freq, dauer[, laut[, kan]]
WAVE kan,wave()

fraq, dauer, laut, kan: iexp
wave(): Integerfeld (%, & oder |) oder sin()

Mit Hilfe der Befehle SOUND und WAVE werden die vier Tongeneratoren des Amiga gesteuert. Der erste Parameter des SOUND-Befehls bestimmt die Frequenz des Tones gemäß folgender Tabelle:

Note	Frequenz in Hertz
C	261,6
D	293,7
E	329,6
F	349,2
G	392,0
A	440,0
H	493,9

Für jede höhere Oktave verdoppelt sich die Frequenz, für niedrigere Oktaven wird die Frequenz jeweils halbiert.

Der zweite Parameter bestimmt die Dauer des Tones. Diese 16-Bit-Zahl gibt an, wie oft die Hüllkurve für diesen Ton wiederholt werden soll.

Der Parameter „laut" kann die Lautstärke im Bereich von 0 bis 255 variieren. Fehlt „laut", so wird die mittlere Lautstärke (127) benutzt.

Der letzte Parameter („kan") legt fest, über welchen Tonkanal der Ton ausgegeben wird. Dabei sind 0 und 3 für den linken, 1 und 2 für den rechten Lautsprecher zuständig. Voreingestellt. ist „0".

Mit dem Befehl WAVE können, für jeden der vier Kanäle getrennt, Hüllkurven bestimmt werden. Voreingestellt ist eine Sinuskurve, die auch durch das Schlüsselwort SIN() erreicht werden kann. Ansonsten stehen im Integerfeld wave%() 256 Byte-Werte, die die Hüllkurvenform bestimmen. Dieses Feld kann sowohl ein Langwort-Integerfeld (%), ein Wort Integerfeld (&) als auch ein Byte-Integerfeld (|) sein.

6 – Disketten- und Dateiverwaltung

Dateispezifikation

Im Folgenden werden die Befehle erklärt, die der Dateien-Organisation dienen. Dazu muß der Aufbau von Dateispezifikationen nach den Regeln des hierarchischen File-Systems bekannt sein. Eine Dateispezifikation besteht aus drei Teilen: der Laufwerksangabe, dem Dateinamen und der Dateikennung. Die Laufwerksangabe enthält die Laufwerkskennung (DF0 ...), gefolgt von einem Doppelpunkt. Der Dateiname darf maximal 30 Zeichen enthalten.

Zur weiteren Untergliederung von Dateien gibt es Verzeichnisse (auch Ordner genannt). Diese Verzeichnisse können in verschiedenen Ebenen angelegt werden; die niedrigste Ebene wird auch Wurzelverzeichnis genannt. Ausgehend von diesem Wurzelverzeichnis kann die Verzweigung in weitere Unterverzeichnisse erfolgen. Ein Verzeichniseintrag kann demnach aus den folgenden drei Teilen bestehen:

- Laufwerksangaben (Diskettenname)
- Namen der Verzeichnisse und Unterverzeichnisse
- Dateinamen

Diese Teile werden durch Schrägstriche „ / " (Slash) getrennt. Namen für Verzeichnisse haben das gleiche Format wie Dateinamen. Somit ergibt sich der Zugriffspfad für eine Datei durch Aneinanderfügen der bisher vorgestellten Elemente. Zunächst die Laufwerksangabe, gefolgt von den Namen der Verzeichnisse, und dann der Dateiname, z.B.: DF0:TEXTE/HANDBUCH/KAPITEL_1. In diesem Beispiel ist der Zugriffspfad also:

DF0:	Laufwerksangabe
TEXTE	Verzeichnis TEXTE
HANDBUCH	Unterverzeichnis HANDBUCH
KAPITEL_1	Dateiname KAPITEL_1

Zusätzlich stehen zwei Sonderzeichen zur Verfügung, die eine leichtere Dateiauswahl ermöglichen. Sie können innerhalb von Dateinamen und deren Kennungen stehen. Es sind dies das Fragezeichen und das Doppelkreuz (#). Das Doppelkreuz in Kombination mit dem Fragezeichen (#?) ermöglicht die Belegung dieser Position durch jedes zulässige Zeichen, dessen ASCII-Code größer als 32 ist. Es ist auch möglich, diese Kombination an mehreren Stellen der Dateispezifikation zu verwenden oder es mehrmals aufeinanderfolgen zu lassen.

Im Folgenden wird der Umgang mit einer sogenannten Random-Access-Datei (Datei mit wahlfreiem Zugriff) erläutert. Dabei sind zwei Begriffe besonders wichtig: Datensatz und Datenfeld.

Ein Datensatz stellt eine logische Zusammenfassung von Daten dar, z.B. der Datensatz „adressen". Dieser Datensatz kann in Datenfelder unterteilt werden, z.B. „name" und „strasse". Analog werden die Begriffe Datensatz- und Feldlänge verwendet. Sie geben die Länge des Datensatzes bzw. der Datenfelder in Bytes an.

Der wesentliche Unterschied zwischen einer Random-Access-Datei und einer sequentiellen Datei besteht im Datenzugriff. Bei einer sequentiellen Datei muß die gesamte Datei geladen werden, um auf einen Datensatz zugreifen zu können. Jedoch kann ein Datensatz aus einer Random-Access-Datei gelesen werden, ohne die gesamte Datei einlesen zu müssen. Dies ist besonders bei sehr großen Dateien nützlich.

Dieser Vorteil wird mit einem größeren Speicherplatzbedarf auf der Diskette erkauft, denn eine Datei mit wahlfreiem Zugriff arbeitet mit festen Datensatz- und Feldlängen. Unabhängig von der Länge der Daten werden diese Satz- und Feldlängen abgespeichert.

CHDIR name$
DIR$(0)
DFREE(0)

name$: sexp

DIR$(n) ermittelt den aktuellen Zugriffspfad. Dabei wird der gesamte Pfad inklusive Laufwerksbezeichnung angegeben. „0“ ist ein Dummy-Parameter und hat keine Funktion. DFREE liefert den freien Speicherplatz des aktuell angewählten Laufwerks in Bytes.

CHDIR legt das aktuelle Verzeichnis fest. Dabei ist der Stringausdruck name$ gleich dem gewünschten Zugriffspfad. Für name$ gelten die selben Konventionen wie z.B. beim Pfadwechsel in CLI:

df0:	Wechselt in das Hauptverzeichnis der Diskettenstation 0.
dh0:devs/printers	Wechselt in ein angegebenes Verzeichnis (devs/printers) der Harddisk 0.
/	Es wird in das übergeordnete Verzeichnis gesprungen. Vorsicht, im Hauptverzeichnis gibt es kein weiteres übergeordnetes Verzeichnis.
:	Wechselt in das Hauptverzeichnis des aktuellen Laufwerks.
:devs	Wechselt in den Ordner „devs“, der im Hauptverzeichnis angelegt ist.

Beispiel:

```
CHDIR ":"
CHDIR "TEXTE_DOC/HANDBUCH"
CHDIR "ANHANG"
CHDIR ":"
```

Wechselt zum Wurzelverzeichnis des aktuellen Laufwerks. In der zweiten Zeile wird zum Unterordner HANDBUCH des Verzeichnisses TEXTE_DOC gewechselt. In der dritten Zeile wird weiter in das Unterverzeichnis ANHANG (TEXTE_DOC/HANDBUCH/ANHANG) verzweigt.

DIR p$ [TO name$]
FILES p$ [TO name$]

p$, name$: sexp

Die Befehle DIR und FILES dienen der Ausgabe von Verzeichnissen. Bei der Verwendung von DIR werden alle Dateien eines bestimmten Laufwerks standardmäßig auf dem aktuellen Fenster ausgegeben. Dabei wird der gewünschte Zugriffspfad im String-Ausdruck p$ angegeben. Die Verwendung von Wildcards ist allerdings nicht möglich. Gleiches gilt für den FILES-Befehl, jedoch werden dabei zusätzlich Länge, Uhrzeit und Datum der betreffenden Dateien aufgeführt. Darüberhinaus sind auch alle Ordner in diesem Verzeichnis angegeben. Order sind mit einem führenden Stern und Dateien mit einem führenden Leerzeichen gekennzeichnet.

Für DIR und FILES ist optional die Angabe des Zusatzes TO name$ möglich. Die Verwendung dieses Zusatzes ermöglicht die Umleitung des Inhaltsverzeichnisses auf eine Datei bzw. ein Peripheriegerät. Der String-Ausdruck name$ enthält dabei den Namen einer Datei bzw. eine Peripheriekennung.

Beispiele:

```
DIR    "DF0:TEXTVERARBEITUNG"
DIR    "DF0:TEXTVERARBEITUNG" TO "DF1:HANDBUCH/INHALT.ASC"
DIR    "DF0:" TO "PRT:"
```

In der ersten Zeile werden alle Dateinamen aus dem Ordner „TEXTVERARBEITUNG“ in Laufwerk DF0 auf dem Bildschirm ausgegeben. Die zweite Zeile bewirkt die Umleitung des Inhaltsverzeichnisses auf eine Datei „INHALT.ASC“ im Laufwerk DF1. Die Ausgabe auf den Drucker geschieht in der dritten Zeile. In der vierten Zeile werden alle Dateinamen mit der Kennung „.DOC“ auf den Drucker umgeleitet.

EXIST(name$)

name$: sexp

Mit Hilfe von EXIST kann man feststellen, ob eine Datei oder ein Ordner vorhanden ist. Dazu wird im String-Ausdruck name$ der Zugriffspfad und der Name dieser Datei angegeben. Die Funktion liefert TRUE (–1) zurück, wenn die Datei oder der Ordner existiert, sonst FALSE (0).

Beispiel:

```
OPEN "U",#1,"TESTTEXT"
PRINT #1,"BEISPIEL"
CLOSE #1
PRINT EXIST("TESTTEXT")
PRINT EXIST("TEST.DOC")
```

In den ersten drei Zeilen wird eine Datei mit dem Namen „TESTTEXT“ eröffnet. Die vierte Zeile überprüft das Vorhandensein der Datei „TEXTTEXT“, es wird TRUE zurückgegeben. In der letzten Zeile ergibt die Überprüfung FALSE, da die Datei nicht existiert.

OPEN modus$,#n,name$ [anz]

modus$, name$: sexp
anz, n: iexp

OPEN öffnet einen Datenkanal zu einer Datei oder zu einem Peripheriegerät. Dabei bestimmt der String-Ausdruck „modus$" eine der folgenden Zugriffsmöglichkeiten:

O (output)	Öffnet eine Datei zum Schreiben. Dazu wird sie ggf. neu eingerichtet bzw. werden bestehende Daten gelöscht.
I (input)	Öffnet eine Datei zum Lesen.
A (append)	Ermöglicht das Anhängen von Daten an eine bestehende Datei. Dabei ist der Datenzeiger auf das Ende der Datei gerichtet.
U (update)	Öffnet eine bereits bestehende Datei zum Schreiben und Lesen.
R (random access)	Öffnet eine Datei zum wahlfreien Schreiben und Lesen. (Eine Beschreibung zu diesem Dateityp wird unter dem FIELD-Befehl gegeben.)

Der numerische Ausdruck n enthält die Kanalnummer und kann Werte von 0 bis 99 annehmen. Diese Kanalnummer muß beim Arbeiten mit der Datei angegeben werden. Hierbei kann das #-Zeichen vor der Kanalangabe entfallen. Der String-Ausdruck name$ enthält den Datei-Zugriffspfad. Als Dateiname kann auch eine Peripheriekennung angegeben werden. Achten Sie allerdings darauf, daß der Modus in „modus$" mit dem gewählten Gerät auch durch-führbar ist. Der numerische Ausdruck „anz" wird nur bei Dateien mit wahl-freiem Zugriff (random access) ausgewertet. Er enthält die Länge eines Datensatzes.

Ein spezieller Dateiname ist CON:, hier erfolgt die ganz normale fensterorientierte Ein- und Ausgabe. Vom Betriebssystem werden einige Namen für spezielle Geräte verwendet: PRT: (Drucker) mit Steuerzeichenumsetzung, PAR: (paralelle Schnittstelle) ohne Steuerzeichenumsetzung, SER: (blockorientierte serielle Schnittstelle), AUX: (Zeichenorientierte serielle Schnittstelle, PIPE: (Inter-Programm-Schnittstelle), SPEAK: (Sprachausgabe), CON:x/y/b/h/Titel (Konsolfenster des DOS mit Angabe von x-, y-Position, Breite, Höhe und Titelzeile) usw.

LOF(#n)
LOC(#n)
EOF(#n)
CLOSE [#n]
TOUCH [#] n

anz, n: iexp

Die Funktionen LOF (length of file), LOC (location) und EOF (end of file) können nur auf zuvor mit OPEN geöffnete Dateien angewandt werden. Alle drei Funktionen haben den numerischen Ausdruck „n" gemeinsam. Er gibt die Kanalnummer der Datei an, auf die sich die jeweilige Funktion bezieht. LOF liefert die Länge einer Datei in Bytes. LOC gibt die aktuelle Position des Datenzeigers auch in Bytes an (siehe auch SEEK). EOF gibt an, ob der Datenzeiger auf das Ende einer Datei zeigt (bzw. die Datei vollständig gelesen wurde). Wenn der Datenzeiger auf das Dateiende zeigt, wird TRUE (–1) zurückgegeben, sonst FALSE (0).

CLOSE schließt einen zuvor mit OPEN geöffneten Datenkanal zu einer Datei oder zu einem Peripheriegerät. Dabei enthält der numerische Ausdruck „n" die Nummer des zu schließenden Kanals. Ohne besondere Kanalangabe werden alle geöffneten Dateien geschlossen.

TOUCH aktualisiert die Zeitangabe einer Datei, das heißt, Datums- und Zeiteintrag einer geöffneten Datei werden auf die Systemdaten umgestellt.

Beispiele:

```
OPEN "O",#1,"testtext"
FOR i%=1 TO 20
  PRINT #1,STR$(i%)
NEXT i%
CLOSE #1
FILES "testtext"
DELAY 20                    ! 20 Sekunden Pause
OPEN "u",#1,"testtext"
TOUCH #1
CLOSE #1
FILES "testtext"
```

Im Beispiel wird die Datei „testtext" unter der Kanalnummer 1 zum Schreiben geöffnet und geschlossen. Anschließend werden auf dem Bildschirm die Daten dieses Files ausgegeben, nach 20 Sekunden aktualisiert und noch einmal ausgegeben.

```
OPEN "I",#1,"testtext"
PRINT "    Dateilänge: ";LOF(#1)
PRINT
PRINT "    Daten","Position des Datenzeigers"
DO UNTIL EOF(#1)
  INPUT #1,a$
  PRINT "    ";a$,LOC(#1)
LOOP
CLOSE #1
```

Das Beispiel öffnet dieselbe Datei zum Lesen. Zunächst wird die Länge der Datei mittels LOF ausgegeben. Anschließend werden Inhalt der Datei und die zugehörige Datenzeigerposition auf dem Bildschirm ausgegeben. Die Abbruchbedingung für die Schleife wird mit EOF festgelegt. Das Einlesen der Daten geschieht (unabhängig von der zuvor ermittelten Dateilänge) durch Überprüfung des Datenzeigers. Dabei wird die Schleife so oft durchlaufen, bis die Funktion EOF den Wert TRUE (–1) übergibt.

NAME alt$ AS neu$
RENAME alt$ AS neu$
KILL name$

alt$, neu$, name$: sexp

NAME benennt eine Datei mit dem Zugriffspfad alt$ in die Datei mit dem Zugriffspfad neu$ um. Dabei wird der Inhalt der Datei nicht verändert. Die Dateien alt$ und neu$ müssen sich auf demselben Laufwerk befinden. RENAME entspricht dem Befehl NAME.

KILL löscht eine Datei, deren Zugriffspfad im String-Ausdruck „name$" angegeben wird.

Beispiel:

```
OPEN "O",#1,"testtext"
PRINT #1,"Beispiel"
CLOSE #1
'
NAME "testtext" AS "beispieltext"
DIR
KILL "beispieltext"
DIR
```

Im ersten Beispiel wird die Datei „testtext" zum Schreiben geöffnet. Anschließend wird diese Datei in „beispieltext" umbenannt und schließlich gelöscht. Dies wird durch das Ausgeben des Inhaltsverzeichnisses überprüft. Nach NAME bzw. KILL werden die veränderten Verzeichnisse auf dem Bildschirm angezeigt.

BLOAD / BSAVE / BGET / BPUT

BLOAD name$ [,adr]
BSAVE name$,adr,anz
BGET #n,adr,anz
BPUT #n,adr,anz

name$: sexp
n, adr, anz: iexp

Mit BSAVE kann man einen Speicherbereich auf Diskette (RAM-Disk, Festplatte usw.) speichern. Mit BLOAD kann er wieder geladen werden. Der numerische Ausdruck adr gibt die Anfangsadresse des Speicherbereiches an. Ohne Angabe von adr wird die bei BSAVE zuletzt genannte Adresse benutzt. Bei BSAVE muß zusätzlich anz (die Länge der Datei in „name$") angegeben werden. Der Parameter „name$" ist der Name der zu ladenden oder zu speichernden Datei. Für „name$" gelten die unter DIR beschriebenen Regeln des hierarchischen Dateisystems.

BSAVE, BLOAD können nur gesamte Dateien unter ihrem Namen ansprechen. Im Gegensatz dazu greifen BPUT und BGET auf Dateien über deren Kanalnummer n zu. So ist es mit BGET und BPUT auch möglich, Teile einer Datei (auch mehrfach) zu laden oder zu speichern.

Beispiele:

```
DATA 1,2,3,4,5,6,7,8,9,0
'
a%=MALLOC(12,0)
FOR i%=0 TO 9
  READ b%
  POKE a%+i%,b%
NEXT i%
BSAVE "DATABLOCK",a%,10
b%=MFREE(a%,12)
```

Speichert den Datenbereich unter dem Namen DATABLOCK auf das aktuelle Diskettenlaufwerk.

INP(#n)
OUT #n,a [,b,c,...]

n, a, b, c,...: iexp

INP(#n) liest ein Byte aus einer zuvor geöffneten Datei. Analog übergibt OUT ein Byte an eine Datei. Dabei enthält der numerische Ausdruck „n“ die Kanalnummer (0 bis 99), unter der die entsprechende Datei angesprochen wird.

Beispiel:

```
OPEN "O",#1,"testtext"
OUT #1,128
CLOSE #1
'
OPEN "I",#1,"testtext"
a=INP(#1)
CLOSE #1
PRINT a
```

Im ersten Beispiel wird eine Datei zum Schreiben geöffnet und ein Byte geschrieben, welches im zweiten Teil des Beispiels in die numerische Variable „a“ eingelesen und auf den Bildschirm ausgegeben wird. Dabei hat „a“ den Wert 128.

INPUT$(anz [,#n])

n, anz: iexp

INPUT$ liest eine Zeichenkette, die aus anz Zeichen besteht, von der Tastatur. Optional bewirkt die Angabe einer Kanalnummer „n“ (0 bis 99) das Einlesen von Zeichen aus einer Datei. In beiden Fällen gibt der numerische Ausdruck „anz“ an, wie viele Zeichen gelesen werden.

Beispiel:

```
OPEN "O",#1,"version.dat"
PRINT #1,"GFA-BASIC, Version 3.0"
CLOSE #1
'
OPEN "I",#1,"version.dat"
v$=INPUT$(9,#1)
CLOSE #1
PRINT v$
PRINT "Geben Sie bitte die Versionsnummer an: ";
PRINT INPUT$(3)
```

Im ersten Beispiel wird die Datei „version.dat“ geöffnet und eine Meldung in die Datei geschrieben. Der zweite Teil des Beispiels liest die ersten neun Zeichen dieser Datei in die String-Variable v$ ein und gibt sie anschließend auf den Bildschirm aus. Danach erscheint eine Meldung, und es werden drei Zeichen von der Tastatur eingelesen und im Fenster ausgegeben.

INPUT #n,var1 [,var2,var3,...]
LINE INPUT #n,a1$ [,a2$,a2$,...]

n: iexp
a1$, a2$, a3$: sexp
var1, var2, var3: avar oder svar

INPUT #n ermöglicht das Einlesen von Daten aus einer Datei. Dabei können einzelne Werte oder Variablenlisten eingelesen werden, wobei die Variablen durch Kommata getrennt werden. Diese Befehle entsprechen INPUT bzw. LINE INPUT, nur wird hier (meist) nicht von der Tastatur gelesen.

Beispiel:

```
OPEN "I",#1,"TEXTE"
INPUT #1,a$,b$
LINE INPUT #1,c$
CLOSE #1
PRINT a$
PRINT b$
PRINT c$
```

Liest drei Strings aus einer Datei, die vorher geöffnet werden muß und gibt sie auf den Bildschirm aus.

PRINT #n,ausdruck
PRINT #n,USING form$,ausdruck
WRITE #n,ausdruck

n: iexp
form$: sexp
ausdruck: aexp oder sexp oder deren Kombinationen

PRINT #n gibt Daten auf einen Datenkanal aus. PRINT #n, USING erlaubt die formatierte Ausgabe auf einen Datenkanal. In beiden Fällen steht dabei n für die Kanalnummer (0 bis 99) der entsprechenden Datei. Ansonsten wie PRINT, PRINT USING und WRITE. Allerdings ist PRINT #n,AT(,) nicht möglich.

Der Befehl WRITE dient in erster Linie der platzsparenden Datenspeicherung in sequentielle Dateien und ist auf das Einlesen von Dateien mit dem INPUT-Befehl zugeschnitten. Die Ausdrücke werden durch Kommata getrennt. Zeichenketten müssen in Anführungszeichen eingeschlossen werden.

Beispiele:

```
OPEN "O",#1,"TEXTE"
b$="wort"
PRINT #1,"Test",a$
PRINT #1,"GFA-","BASIC"
CLOSE #1
```

Schreibt drei Strings in die eingerichtet Datei.

```
OPEN "O",#1,"TEST.DAT"
WRITE #1,"Version",3,".O"
CLOSE #1
'
OPEN "I",#1,"TEST.DAT"
INPUT #1,v1$,v2$,v3$
CLOSE #1
'
PRINT v1$+v2$+v3$
```

Schreibt Daten durch Kommata getrennt in die Datei TEST.DAT, liest anschließend die Daten mit INPUT wieder ein und gibt sie auf den Bildschirm aus.

STORE #i,x$() [,n[TO z]]
RECALL #i,x$(),n[TO z],x

i, n, z: iexp
x$(): String-Array
x: Variable, mindestens 32 Bit

Der Befehl STORE dient zum Abspeichern eines String-Feldes als Text-File (mit Trennung durch LF). Es wird das komplette String-Array über den geöffneten Kanal i ausgegeben. Der wahlfreie Parameter n kann angeben, wieviele Elemente des String-Arrays geschrieben werden sollen. Mit n TO z wird ein Teilfeld gespeichert (bzw. gelesen).

Der Befehl RECALL dient zum schnellen Einlesen eines String-Feldes aus einem Text-File. Es werden n Zeilen des Text-Files in das String-Array eingelesen. Wenn n zu groß für die Dimensionierung des String-Arrays ist, so wird die einzulesende Anzahl automatisch begrenzt (n=–1 liest das gesamte Array).

Wird während des Lesens das File-Ende (EOF) erreicht, so wird das Einlesen ohne Fehlermeldung abgebrochen. In jedem Fall enthält die Variable x am Ende die tatsächlich gelesene Anzahl Strings.

Beispiele:

```
DIM A$(1000)
FOR i%=0 TO 499
  a$(i%)=STR$(RND)        !irgendetwas
NEXT i%
OPEN "O",#1,"testfile.text"
STORE #1,a$(),500
CLOSE #1
'
ERASE a$()
DIM b$(2000)
OPEN "I",#1,"testfile.text"
RECALL #1,b$(),-1,n
CLOSE #1
PRINT n
```

Es wird die Anzahl der gelesenen Textzeilen ausgegeben. (Hier also 500.)

```
PRINT "Zeilenzähler"
DIM a$(1000)
DO
  FILESELECT "Zeilen zählen","OK","",f$
  EXIT IF f$=""
  lc%=0
  OPEN "I",#1,f$
  DO
    RECALL #1,a$(),-1,x%
    ADD lc%,x%
  LOOP WHILE x%
  CLOSE #1
  PRINT f$;" enthält ";lc%;" Zeilen"
LOOP
```

Dieses Programm zählt die Zeilen in Text-Files.

SEEK #n,pos
RELSEEK #n,anz

n, anz, pos: iexp

Die Befehle SEEK und RELSEEK erlauben die Positionierung des Datenzeigers. Dadurch läßt sich ein indexsequentieller Dateizugriff realisieren. Dabei enthält der numerische Ausdruck „n" die Kanalnummer einer zuvor mit OPEN geöffneten Datei. Beide Befehle können nur bei Dateien, nicht aber bei Peripheriegeräten verwendet werden. Der Datenzeiger gibt an, welches Byte einer Datei zuletzt gelesen oder geschrieben wurde. Außer beim Modus „A" (Datenanfügen an eine bestehende Datei) hat der Datenzeiger beim Öffnen einer Datei den Wert 0. Die Schreib- und Lesebefehle werden mit dem ersten Byte begonnen, auf das der Datenzeiger gerichtet ist.

Der absoluten Positionierung des Datenzeigers dient der SEEK-Befehl. Dazu wird der Datenzeiger auf das in „pos" angegebene Byte gerichtet. Die relative Zeigerpositionierung erfolgt mit dem RELSEEK-Befehl. Dabei wird der Datenzeiger von seiner aktuellen Position um den in „anz" angegebenen Wert verschoben. RELSEEK ist meistens schneller! Die numerischen Ausdrücke „anz" und „pos" dürfen nur Werte von 0 bis zur entsprechenden Dateilänge annehmen. Bei positiven Werten wird der Datenzeiger in Richtung Dateiende und bei negativen Werten in Richtung Dateianfang bewegt. Bei der Angabe von 0 wird der Datenzeiger auf den Dateianfang gerichtet.

Beispiel:

```
OPEN "O",#1,"X.X"
PRINT #1,STRING$(20,42)
SEEK #1,10
PRINT #1,"#";
RELSEEK #1,-5
OUT #1,33,48
CLOSE #1
OPEN "I",#1,"X.X"
LINE INPUT #1,a$
PRINT a$
CLOSE #1
```

Ausgabe: ******!0**#*********

FIELD #n,anz AS satz$ [, anz AS satz$, anz AS satz$,...]
FIELD #n,anz AT(x) [,anz AT(x) [,anz AT(x) [,,...]

n, anz,x: iexp
satz$: svar, jedoch keine Feldvariable

Der Befehl FIELD AS unterteilt Datensätze in Felder. Der numerische Ausdruck n ist die Nummer des Datenkanals (0 bis 99) einer mit OPEN geöffneten Datei. Der ganzzahlige Ausdruck anz bestimmt die Feldlänge. Die String-Variable satz$ nimmt ein Feld eines Datensatzes auf. Soll der Datensatz in mehrere Felder aufgeteilt werden, müssen die Befehlsteile (anz AS satz$) durch Kommata getrennt werden. Die Summe der einzelnen Feldlängen muß der Datensatzlänge entsprechen, sonst wird eine entsprechende Fehlermeldung ausgegeben. Damit die einzelnen Datensätze keine andere Länge haben als im FIELD-Befehl festgelegt, ist es zweckmäßig, die Befehle LSET und RSET oder MID$ zu benutzen.

Mit Hilfe von AT können z.B. numerische Variablen in eine „R"-Datei (Random Access) geschrieben werden, ohne daß sie in Zeichenketten überführt werden müssen. Die geschieht, indem in den Klammern ein Zeiger auf die abzuspeichernde numerische Variable und vor AT die Anzahl der Bytes, die ab dieser Adresse gelesen und abgespeichert werden soll, steht. Es können auch beliebige andere Adressen angegeben werden. Zum Beispiel:

```
FIELD #1,4 AT(*a%),2 AT(*b&),8 AT(*c#)
```

Außerdem sind beliebige Kombinationen von AS und AT möglich, z.B.:

```
FIELD #2,4 AS a$,2 AT(*b&),8 AT(*c#),6 AS d$
```

GET #n [,satz]
PUT #n [,satz]
RECORD #n,satz

n, satz: iexp

GET liest einen Datensatz aus einer „R"-Datei. Analog speichert PUT einen Datensatz in eine solche Datei. Dabei ist n die Nummer (0 bis 99) eines mit OPEN geöffneten Datenkanals. Der optionale Parameter satz enthält einen Wert zwischen 1 und der Anzahl der Datensätze innerhalb der entsprechenden Datei. Dieser gibt die Nummer des zu lesenden oder zu speichernden Datensatzes an. Ohne Angabe von satz wird der jeweils nächste Datensatz gelesen oder gespeichert.

Mit RECORD wird nur die nächste Satznummer für PUT oder GET gesetzt. Nach RECORD #1,15 liest GET #1, also Satz Nr. 15.

Achtung: Eine Datei kann immer nur um einen Datensatz verlängert werden, gegebenenfalls bei Anlegen in einer Schleife, mehrere Datensätze anlegen.

Beispiele:

```
OPEN "R",#1,"privat.rdm",62
FIELD #1,24 AS name$,24 AS strasse$,2 AT(*plz&),12 AS ort$
'
FOR i%=1 TO 3
  INPUT "Name   : ";n$
  INPUT "Strasse : ";s$
  INPUT "PLZ     : ";plz&
  INPUT "Ort     : ";o$
  LSET name$=n$
  LSET strasse$=s$
  LSET ort$=o$
  PUT #1,i%
  CLS
NEXT i%
'
CLOSE #1
```

Zunächst wird eine Random-Access-Datei (Modus „R“) mit einer Satzlänge von 62 Bytes eröffnet. Mit Hilfe des FIELD-Befehls wird ein Datensatz in zwei Datenfelder mit je 24, ein Feld mit zwei und ein Datenfeld mit zwölf Bytes unterteilt. Die Summe der Datenfelder (24+24+2+12) ergibt wieder 62 (also die beim OPEN-Befehl angegebene Datensatzlänge). Anschließend werden die Adreßdaten eingegeben und linksbündig in die entsprechenden Variablen eingesetzt. Danach wird der gesamte Datensatz in die Datei geschrieben.

```
OPEN "R",#1,"privat.rdm",62
FIELD #1,24 AS name$,24 AS strasse$,2 AT(*plz&),12 AS ort$
'
FOR i%=1 TO 3
  GET #1,i%
  PRINT "Datensatznummer: ";i%
  PRINT "Name    : ";name$
  PRINT "Strasse : ";strasse$
  PRINT "PLZ     : ";plz&
  PRINT "Ort     : ";ort$
NEXT i%
'
CLOSE #1
```

Hierbei wird die Datei mit wahlfreiem Zugriff „PRIVAT_1.RDM“ geöffnet und es werden drei Datensätze gelesen.

7 – Programmsteuerung

Schleifen

GFA-BASIC 3.0 verfügt über eine ungewöhnlich große Auswahl an Schleifentypen. Normalerweise unterscheidet man abweisende und nicht abweisende Schleifen. Abweisende Schleifen sind solche, bei denen die Abbruchbedingung vor dem Eintritt in die Schleife geprüft wird, während diese Bedingung bei nicht abweisenden Schleifen am Ende des Schleifenkörpers überprüft wird. Die Konsequenz davon ist, daß nicht abweisende Schleifen mindestens einmal durchlaufen werden.

GFA-BASIC kennt zunächst einmal zwei der üblichen nicht abweisenden Schleifentypen, nämlich FOR NEXT- und REPEAT UNTIL-Schleifen. Ein dritter Standardschleifentyp ist die abweisende WHILE WEND-Schleife. Als vierter Schleifentyp steht DO LOOP zur Verfügung. Bei diesem Typ handelt es sich eigentlich nur um eine Endlosschleife ohne Abbruchbedingung, in GFA-BASIC kann dieser Befehl jedoch sehr variabel eingesetzt werden. Sowohl hinter DO als auch hinter LOOP können die Erweiterungen WHILE und UNTIL gesetzt werden, so daß sich z.B. auch Schleifen konstruieren lassen, die am Schleifenanfang und -ende eine logische Bedingung abfragen.

In jede der hier genannten Schleifentypen lassen sich darüber hinaus noch beliebig viele Abbruchbedingungen in den Schleifenkörper einbauen (EXIT IF).

In GFA-BASIC 3.0 erhalten Unterprogramme, wie in jeder modernen Programmiersprache, Namen. Diesen Unterprogrammen können Parameter übergeben werden, sie können auch über VAR-Parameter auf verschiedene Variablen in verschiedenen Aufrufen angewandt werden.

Variablen können an Unterroutinen als Wert (call by value) übergeben werden. Mit Hilfe der Anweisung VAR können sie aber auch selbst übergeben werden (call by reference), so daß sie von der Unterroutine auch verändert werden können, ohne daß ihr globaler Variablenname benutzt werden muß.

Hier gibt es zunächst die Möglichkeit der Vereinbarung lokaler Variablen (LOCAL), die nur in einem Teil des Programms bekannt sind. D.h. es muß keine Rücksicht auf mögliche Namenskollisionen mit den aufrufenden Prozeduren bzw. Funktionen genommen werden.

Mit Hilfe von DEFFN kann man einzeilige Funktionen definieren, die später mit FN unter einem Namen angesprochen werden können. Es sind auch mehrzeilige Funktionen möglich (FUNCTION). Diese stellen grundsätzlich nur eine Sonderform der Prozeduren dar und liefern ein Ergebnis (über RETURN) zurück.

IF bed [THEN]
ELSE
ENDIF

bed: bexp

Diese Kommandos ermöglichen es, einen oder mehrere Befehle nur dann wirksam werden zu lassen, wenn eine logische Bedingung erfüllt ist. Folgendes Beispiel soll dies erläutern:

```
IF a=1 THEN
  PRINT "a ist gleich 1"
  b=2
ENDIF
```

In diesem Fall ist a=1 die logische Bedingung. Die Befehle in den Zeilen zwischen IF und ELSE werden nur abgearbeitet, wenn diese logische Bedingung wahr ist. Ist sie unwahr, so wird der Programmlauf hinter dem Befehlswort ENDIF fortgesetzt. Das Kommando THEN wird nicht beachtet, es genügt also auch die Formulierung: IF a=1 statt IF a=1 THEN.

Etwas komplexer ist folgende Konstruktion:

```
IF a=1
  PRINT "a ist gleich 1"
ELSE
  PRINT " a ist nicht gleich 1,"
  PRINT " sondern gleich ";a
ENDIF
```

In diesem Fall werden die Befehle zwischen IF und ELSE abgearbeitet, wenn die logische Bedingung hinter IF wahr ist. Anschließend fährt das Programm hinter dem Kommando ENDIF fort. Ist die Bedingung hinter IF aber nicht erfüllt, so werden die Befehle zwischen ELSE und ENDIF wirksam. Anschließend wird die Programmausführung wieder hinter ENDIF fortgesetzt. Logisch wahr ist jeder numerische Ausdruck, der ungleich 0 ist.

Beispiel:

```
x=1
IF x
  PRINT "x ist wahr"
ENDIF
INPUT y
IF x=9 OR ODD(y)
  PRINT "y ist eine ungerade Zahl"
ELSE
  PRINT "y ist eine gerade Zahl"
ENDIF
```

Zunächst erscheint der Text „x ist wahr". Dann wird eine Zahl abgefragt. Da x nicht 9 sein kann, erscheint der Text „y ist eine ungerade Zahl", wenn Sie eine ungerade Zahl eingegeben haben, andernfalls wird „y ist eine gerade Zahl" auf den Monitor geschrieben.

ELSE IF bed

bed: bexp

Der Befehl ELSE IF ermöglicht es, verschachelte IF-Anweisungen übersichtlicher darzustellen. Das folgende Beispiel reagiert auf das Drücken von Tasten. Bei S, L oder E wird ein Platzhalter für eine Speicher-, Lade- oder Eingaberoutine aufgerufen. In allen anderen Fällen erscheint der Text „Unbekanntes Kommando". Die verschachtelte Version dieses Beispiels lautet dann:

```
DO
  t$=CHR$(INKEY$)
  '
  IF t$="L"
    PRINT "Laden"
  ELSE
    IF t$="S"
      PRINT "Speichern"
    ELSE
      IF t$="E"
        PRINT "Eingeben"
      ELSE
        PRINT "Unbekanntes Kommando"
      ENDIF
    ENDIF
  ENDIF
  '
LOOP
```

Die Verwendung von ELSE IF erzeugt ein kürzeres Listing mit weniger unübersichtlichen Einrückungen:

```
OPENW 0
DO
  t$=INKEY$
  '
  IF t$="L"
    PRINT "Laden"
  ELSE IF t$="S"
    PRINT "Speichern"
  ELSE IF t$="E"
    PRINT "Eingeben"
  ELSE
    PRINT "Unbekanntes Kommando"
  ENDIF
  '
LOOP
```

Die Abarbeitung solcher Programmstrukturen erfolgt nach folgendem Muster: Ist die Bedingung hinter IF erfüllt (hier t$="L"), so werden die Befehle zwischen IF und dem nächsten ELSE IF abgearbeitet (hier PRINT "Laden") und dann hinter den Befehl ENDIF verzweigt.

Ist eine der Bedingungen hinter den ELSE IF-Befehlen wahr, so werden alle Befehle bis zum nächsten ELSE, ELSE IF oder ENDIF (wenn kein ELSE existiert) abgearbeitet und anschließend zum Befehl hinter ENDIF gesprungen. Ist weder die Bedingung hinter IF noch eine Bedingung hinter ELSE IF wahr, so werden die Befehle zwischen ELSE und ENDIF ausgeführt (sofern ein ELSE existiert).

ON x GOSUB proc1,proc2, ...

x: iexp
proc1, proc2: Prozedurnamen, ohne Parameter

Dieser Befehl verzweigt zur x-ten Prozedur, die in der Liste hinter GOSUB aufgeführt ist. Dabei ist x ein numerischer Ausdruck, dessen Nachkommastellen (sofern vorhanden) ignoriert werden. Ist x kleiner als 1 oder größer als die Anzahl der Prozedurnamen hinter GOSUB, so wird keine Unterroutine aufgerufen. Nach Aufruf der Unterroutine wird das Programm hinter ON x GOSUB fortgesetzt.

Den Prozeduren können bei diesem Befehl keine Parameter übergeben werden.

Beispiel:

```
x=3
ON x GOSUB proc1,proc2,proc3
x=1
ON x+1 GOSUB proc1,proc2,proc3,proc4
```

Es wird zunächst die Prozedur proc3 aufgerufen, dann die Prozedur proc2.

SELECT x
CASE y [TO z] oder CASE y [,z,...]
CASE TO y
CASE y TO
DEFAULT
ENDSELECT
CONT

x, y, z: iexp oder String-Konstante mit einer Maximallänge von vier Zeichen

Der Befehl SELECT ermöglicht Verzweigungen in Abhängigkeit vom Wert des numerischen Ausdrucks x. Folgendes Beispiel steht zur Erläuterung der dadurch entstehenden Programmstruktur:

```
OPENW 0
x=0
SELECT x+2
CASE 1
  PRINT "x ist gleich 1"
CASE 2 TO 4
  PRINT "x ist gleich 2,3 oder 4"
CASE 5,6
  PRINT "x ist gleich 5 oder 6"
DEFAULT
  PRINT "x ist ungleich 1,2,3,4,5 oder 6"
ENDSELECT
```

Es wird „x ist gleich 2,3 oder 4“ auf den Monitor ausgegeben. Zunächst wird der numerische Ausdruck hinter SELECT bestimmt, der die Verzweigungsbedingung ist (hier gleich 2). Anschließend werden die CASE-Anweisungen von oben nach unten durchgegangen und geprüft, ob hinter ihnen der aktuelle Wert der Verzweigungsbedingung zu finden ist. In diesem Beispiel steht hinter der ersten CASE-Anweisung nur die 1. Da die Verzweigungsbedingung gleich 2 ist, springt das Programm zum nächsten CASE.

Hinter dem zweiten CASE steht „2 TO 4". Diese Bedingung ist also erfüllt, wenn das Verzweigungskriterium hinter SELECT Werte zwischen 2 und 4 (einschließlich) annimmt. Im vorliegenden Fall führt dies dazu, daß die Befehle zwischen dem zweiten und dem dritten CASE abgearbeitet werden. Danach wird die Programmausführung hinter dem Kommando ENDSELECT fortgesetzt.

Hinter dem dritten CASE ist eine weitere Variante der Angabe der möglichen Werte des SELECT-Kriteriums zu erkennen. Dort werden die gewünschten Werte in einer Liste, durch Kommata getrennt, aufgezählt. Ist die aktuelle Ausprägung des Verzweigungskriteriums hinter keinem CASE aufgeführt, so werden die Befehle zwischen DEFAULT und ENDSELECT verarbeitet, sofern eine DEFAULT-Anweisung vorhanden ist. Statt DEFAULT kann aber auch OTHERWISE angegeben werden, was der Interpreter automatisch durch DEFAULT ersetzt.

Hinter CASE können jedoch nicht nur numerische Konstanten, sondern auch String-Konstanten mit einer maximalen Länge von vier Zeichen angegeben werden. Dabei kann der hinter CASE angegebene String maximal vier Zeichen enthalten. Bei Angabe nur eines Zeichens wird dessen ASCII-Wert als Verzweigungskriterium benutzt. Werden zwei Zeichen angegeben, so berechnet sich dieser Wert wie folgt:

ASCII-Wert des ersten Zeichens + 255 * ASCII-Wert des zweiten Zeichens. Entsprechend wird bei Angabe von drei bzw. vier Zeichen verfahren.

Beispiele:

```
OPENW 0
raus!=FALSE
REPEAT
  taste%=INKEY$
  SELECT taste%
  CASE "a" TO "z"
    PRINT "der Kleinbuchstabe "+chr$(taste%)+" wurde eingegeben"
  CASE "A" TO "Z"
    PRINT "der Großbuchstabe "+chr$(taste%)+" wurde eingegeben"
  CASE 27
```

```
    raus!=TRUE     !Programmende durch Drücken von <Esc>
  DEFAULT
    PRINT "eine unzulässige Taste wurde gedrückt!"
  ENDSELECT
UNTIL raus!
```

Innerhalb einer Schleife wird die Tastatur abgefragt und je nach gedrückter Taste verzweigt sowie das entsprechende Zeichen ausgegeben bzw. eine unzulässige Eingabe gemeldet. Als Abbruchkriterium für die Schleife dient das Drücken der ESC-Taste.

Das nächste Beispiel illustriert die Bedeutung der CONT-Anweisung, die nur dann eine Wirkung hat, wenn sie vor einem CASE- oder DEFAULT-Befehl steht. Dieser CONT-Befehl darf nicht mit dem gleichnamigen Kommando zur Wiederaufnahme eines unterbrochenen Programmlaufs verwechselt werden.

```
OPENW 0
x=1
SELECT x
CASE 1
  PRINT "x ist gleich 1"
  CONT
CASE 2
  PRINT "x ist gleich 2"
CASE 1,3
  PRINT "x ist gleich 3"
DEFAULT
  PRINT "x ist ungleich 1,2 und 3"
ENDSELECT
```

Es wird „x ist gleich 1“ und „x ist gleich 2“ im Fenster ausgegeben. Das CONT-Kommando sorgt dafür, daß die hinter ihm stehende CASE-oder DEFAULT-Anweisung übersprungen wird. In diesem Beispiel ist der Wert hinter der ersten CASE-Anweisung gleich der Verzweigungsbedingung. Folglich wird der Befehl PRINT "x ist gleich 1" ausgeführt. Anschließend folgt eine CONT-Anweisung, die die Zeile CASE 3 überspringt, obwohl die Verzweigungsbedingung nicht erfüllt, also ungleich 1, ist. Das führt dazu, daß auch der Befehl PRINT "x ist gleich 2" ausgeführt wird.

Beim Erreichen der nächsten CASE- oder DEFAULT-Anweisung erfolgt nun der Sprung hinter ENDSELECT. Dies geschieht auch dann, wenn (wie hier) hinter diesem CASE die aktuelle Ausprägung der Verzweigungsbedingung aufgeführt ist.

```
SELECT INKEY$
CASE "a" TO "g"
  PRINT "a bis g"
DEFAULT
  PRINT "default"
ENDSELECT
```

Wenn Sie eine der Tasten a,b,c,d,e,f oder g drücken, erscheint der Text „a bis g", bei einer anderen Taste (nicht Tastaturumschalttaste) erscheint „default".

Bei CASE können die verschiedenen Möglichkeiten kombiniert werden, es ist also nicht nötig zu schreiben:

```
SELECT a$
CASE "a" TO "z"
  CONT
CASE "A" TO "Z"
  CONT
CASE "ä","ö","ü","ß","Ä","Ö","Ü"
  PRINT "OK"
ENDSELECT
```

sondern es reicht (auch):

```
SELECT a$
CASE "a" TO "Z", "A" TO "Z","ä","ö","ü",...
...
...
```

FOR i= a TO e [STEP s]
(Befehle)
NEXT i
DOWNTO

i: avar
a, e, s: aexp

Die FOR NEXT-Schleife (auch Zählschleife genannt) dient dazu, eine zwischen den Befehlsworten FOR und NEXT stehende Befehlsgruppe (Schleifenkörper) wiederholt abzuarbeiten. Zu diesem Zweck wird die Laufvariable i zunächst mit dem Anfangswert a versehen. Nun werden die Befehle des Schleifenkörpers durchlaufen, bis NEXT erreicht wird. Dort wird die Variable i um den Wert s erhöht, der hinter STEP steht. Ist kein STEP vorhanden, so wird i um den Wert 1 erhöht. Jetzt wird geprüft, ob i den Wert e überschritten hat. Wenn dies der Fall ist, wird der Befehl hinter NEXT angesprungen, wenn nicht, so wird wieder beim ersten Befehl des Schleifenkörpers begonnen. Dieser Vorgang wird wiederholt, bis i größer als e ist.

Eine Konsequenz dieses Ablaufs ist, daß i nach dem Verlassen der Schleife immer gleich dem ersten Wert ist, der das Abbruchkriterium überschritten hat. Der Inhalt der FOR-Schleife wird mindestens einmal durchlaufen.

Statt der STEP-Schrittweite s=–1 kann auch die Anweisung DOWNTO verwendet werden. Im Zusammenhang mit DOWNTO ist die Verwendung von STEP dann nicht mehr möglich.

Als Laufvariable i sollte man Integervariablen verwenden, weil dadurch die Schleifenbefehle schneller abgearbeitet werden können als mit Fließkomma-Laufvariablen. Dies ist natürlich bei nicht ganzzahligen Schrittweiten nicht möglich.

Statt des Befehlswortes NEXT, gefolgt von der Laufvariable (z.B. i%), kann auch endfor i% geschrieben werden, was der Interpreter durch NEXT i% ersetzt.

Beispiele:

```
OPENW 0
FOR i=1 TO 10
  PRINT i'
NEXT i
FOR i=-1 DOWNTO -10
  PRINT i'
NEXT i
```

Schreibt die Zahlen 1 2 3 4 5 6 7 8 9 10 –1 –2 –3 –4 –5 –6 –7 –8 –9 –10 in das Fenster.

```
a$="T*e*s*t*w*o*r*t"
FOR j=1 TO LEN(a$) STEP 2
  PRINT MID$(a$,j,1);
NEXT j
```

Es erscheint „Testwort“ im Fernster.

REPEAT
(Befehle)
UNTIL bed

bed: bexp

Mit REPEAT — UNTIL bed wird eine Anzahl von Befehlen eingerahmt, die ausgeführt werden, bis die logische Bedingung bed wahr ergibt.Wird im Programm die Anweisung REPEAT erreicht, so werden die dahinter stehenden Befehle bis zum Erreichen von UNTIL abgearbeitet. Nun wird geprüft, ob die hinter UNTIL stehende logische Bedingung bed wahr ist. Wenn dies der Fall ist, so werden die Befehle hinter UNTIL ausgeführt. Ergibt bed aber falsch (0), dann springt die Programmausführung wieder zur REPEAT-Anweisung.

Die Befehle zwischen REPEAT und UNTIL werden mindestens einmal abgearbeitet, sofern die Schleife nicht mit EXIT IF oder GOTO verlassen wird. Es handelt sich um eine nicht abweisende Schleife. Statt UNTIL bed kann auch ENDREPEAT bed geschrieben werden, was der Interpreter automatisch durch UNTIL ersetzt.

Beispiele:

```
OPENW 0
REPEAT
UNTIL MOUSEK
```

Wartet auf einen Maustastendruck.

```
i=1
REPEAT
  INC i
  j=SQR(i)
UNTIL i>10 and FRAC(j)=0
PRINT i
```

Es wird die Zahl 16 ausgegeben.

WHILE bed
(Befehle)
WEND

bed: bexp

Die Anweisungen WHILE und WEND können eine Gruppe von Befehlen einschließen, die solange abgearbeitet werden, wie die logische Bedingung bed wahr ist. Stößt GFA-BASIC auf den Befehl WHILE, so wird die dahinter stehende logische Bedingung geprüft. Wenn sie wahr ist, so werden die Befehle zwischen WHILE und WEND ausgeführt. Beim Erreichen von WEND springt das Programm wieder zu WHILE und der Zyklus beginnt von neuem, bis „bed" falsch wird. Statt der WEND-Anweisung kann auch ENDWHILE geschrieben werden, was der Interpreter durch WEND ersetzt.

Beispiel:

```
OPENW 0
WHILE MOUSEK<>2
  PLOT MOUSEX,MOUSEY
WEND
```

Zeichnen mit der Maus, bis die rechte Maustaste gedrückt wird.

DO
(Befehle)
LOOP

Die Befehle DO LOOP erzeugen eine Endlosschleife. Das Programm arbeitet die zwischen DO und LOOP stehenden Befehle ab und springt beim Erreichen von LOOP wieder zu dem Kommando DO zurück.

Die Schleife kann vom Programm nur mit Hilfe von EXIT IF, GOTO oder Programmende-Befehlen verlassen werden. Statt der LOOP-Anweisung kann auch ENDDO geschrieben werden, was der Interpreter durch LOOP ersetzt.

Beispiel:

```
OPENW 0
GRAPHMODE 1
COLOR 1,2,1
DEFFILL 1,3,4
DO
  MOUSE mx,my,mk
  IF mk
    PBOX mx,my,mx+25,my+25
  ENDIF
LOOP
```

Zeichnet bei Maustastendruck ausgefüllte Rechtecke an der aktuellen Maus-Cursor-Position.

DO WHILE bed
DO UNTIL bed
LOOP WHILE bed
LOOP UNTIL bed

bed: bexp

Die Befehle DO und LOOP können um die Zusätze UNTIL und WHILE erweitert werden. Der Schleifenkopf DO WHILE führt dazu, daß das Innere der Schleife nur bearbeitet wird, solange „bed" wahr ist. Beginnt die Schleife mit DO UNTIL, so wird sie nur betreten, wenn die Bedingung „bed" nicht erfüllt ist.

LOOP WHILE führt dazu, daß das Programm zu DO zurückspringt, solange „bed" wahr ist. Für LOOP UNTIL gilt, daß „bed" falsch sein muß, damit der Rücksprung zum Schleifenanfang erfolgt.

Daher sind die Bedingungen bei DO abweisend, bei LOOP nicht abweisend.

```
DO WHILE bed                WHILE bed
.                           .
.            entspricht     .
.                           .
LOOP                        WEND

DO                          REPEAT
.                           .
.            entspricht     .
.                           .
LOOP UNTIL bed              UNTIL bed
```

Die Befehlsvarianten DO, DO WHILE und DO UNTIL können beliebig mit LOOP, LOOP WHILE und LOOP UNTIL kombiniert werden, so daß sich mit diesen Befehlen insgesamt neun Schleifenarten konstruieren lassen.

Beispiele:

```
OPENW 0
DO
LOOP UNTIL MOUSEK
' Wartet auf den Druck einer Maustaste.
DO UNTIL MOUSEK=2
  DO WHILE MOUSEK=1
    LINE 0,0,MOUSEX,MOUSEY
  LOOP
LOOP UNTIL INKEY$="a"
```

Zeichnet bei gedrückt gehaltener linker Maustaste Linien. Wenn die rechte Maustaste oder „a" gedrückt wird, endet das Programm.

```
DO UNTIL EOF(#1)
  INPUT #1,a$
LOOP
```

Liest aus Kanal 1 sequentiell Zeichenketten, bis das Dateiende erreicht wird.

```
WHILE NOT EOF(#1)
  INPUT #1,a$
WEND
```

Konstruktion mit WHILE WEND ist langsamer, da zusätzlich NOT benötigt wird.

EXIT IF bed

bed: bexp

Mit Hilfe von EXIT IF kann aus einer Schleife gesprungen werden, wenn die bool'sche Bedingung „bed" erfüllt ist. Der Schleifentyp ist dabei beliebig wählbar.

EXIT IF kann auch innerhalb von IF-ENDIF und SELECT-ENDSELECT angewendet werden.

Beispiel:

```
DO
  EXIT IF MOUSEK
LOOP
REPEAT
  EXIT IF INKEY$="x"
UNTIL FALSE
```

Beendet das Programm, wenn zunächst eine Maustaste und dann die Taste „x" gedrückt wird.

GOSUB proc [(par1,par2,...)]
PROCEDURE proc [(var1,var2,...)]
RETURN

proc: Name der Prozedur
par1, par2: sexp, aexp
var1, var2: svar,avar

Zwischen den Befehlsworten PROCEDURE und RETURN stehen die Befehle eines Unterprogramms. Hinter PROCEDURE steht der Name des Unterprogramms und eventuell die Liste der zu empfangenden Variablen. Der Aufruf einer PROCEDURE erfolgt unter Angabe ihres Namens am Zeilenanfang, gegebenenfalls gefolgt von entsprechenden Parametern, die in Klammern gesetzt sind. Zur Verdeutlichung, daß es sich dabei nicht um GFA-BASIC-Befehle handelt, kann wahlweise ein „@" oder das Schlüsselwort „GOSUB" vorangestellt werden. Bei Verwechslungsmöglichkeiten mit GFA-BASIC-Befehlen ist dies vorgeschrieben (z.B. @stop, @rem).

Die Übergabeparameter können Konstanten, Variablen und Ausdrücke sein. Von Variablen können nicht nur die Werte, sondern auch die Variablen selbst übergeben werden (siehe VAR).

Wird bei der Programmausführung die Anweisung RETURN erreicht, so springt das Programm zu dem Befehl, der hinter GOSUB steht. Statt des Befehlswortes PROCEDURE kann auch SUB geschrieben werden, statt RETURN auch ENDPROC oder ENDSUB. Dies ersetzt der Interpreter selbst.

Beispiel:

```
OPENW 0
GOSUB slow_print (" ** Handbuch zu    **")
@slow_print("* GFA-BASIC 3.0    *")
slow_print("GFA-SYSTEMTECHNIK")
'
PROCEDURE slow_print(t$)
  LOCAL i%
  FOR i%=1 TO LEN(t$)
    PRINT MID$(t$,i%,1);
    PAUSE 3
  NEXT i%
  PRINT
RETURN
```

Langsames, zeichenweises Ausgeben eines Strings.

```
a=8
@dritte_wurzel(a)
PRINT a
'
PROCEDURE dritte_wurzel(VAR x)
  x=x^(1/3)
RETURN
```

Berechnet die dritte Wurzel von 8 und gibt 2 auf dem Bildschirm aus.

LOCAL var1 [,var2,var3,...]

var1, var2, var3: avar, svar

Es ist mit Hilfe des Befehls LOCAL möglich, den Geltungsbereich von Variablen einzuschränken. Die hinter LOCAL aufgeführten Variablen sind nur in der Prozedur gültig, in der LOCAL steht sowie in allen von dieser Prozedur aufgerufenen Unterprogrammen.

Hinter LOCAL dürfen auch Variablen stehen, die im Hauptprogramm, also global, gültig sind. Diese Variablen können dann in der Unterroutine nicht angesprochen werden, stehen nach dem Verlassen der Prozedur aber wieder unverändert zur Verfügung.

Übergabevariablen einer Prozedur oder Funktion sind immer lokal.

Beispiel:

```
x=2
GOSUB test
PRINT x,y
'
PROCEDURE test
  LOCAL x,y
  x=3
  y=4
RETURN
```

Es werden die Zahlen 2 und 0 ausgegeben.

@funk [(par1,par2,...)]
FUNCTION funk [(var1,var2,...)]
RETURN exp
ENDFUNC

funk:	Name der Funktion
par1, par2:	sexp, aexp
var1, var2:	svar, avar
exp:	sexp, aexp

Zwischen den Befehlsworten FUNCTION und ENDFUNC stehen die Befehle einer Unterroutine (ähnlich PROCEDURE). Dabei steht hinter FUNCTION der Name der Unterroutine und eventuell die Liste der zu übergebenden Variablen. Der Aufruf der Unterroutine erfolgt durch die Angabe des „Klammeraffen" @ oder FN und des Funktionsnamens. Dann folgt gegebenenfalls eine Parameterliste.

Die Übergabeparameter können Konstanten, Variablen und Ausdrücke sein. Von Variablen können nicht nur die Werte, sondern auch die Variablen selbst übergeben werden (siehe VAR).

Wird bei der Programmausführung die Anweisung RETURN erreicht, so wird der in der Funktion ermittelte Wert an der Stelle des Funktionsaufrufes eingesetzt. In einer Funktion kann RETURN auch mehrfach verwendet werden, mit IF o.ä., es ist aber nicht erlaubt, eine Funktion ohne RETURN zu beenden (z.B. durch ENDFUNC).

Ein $-Zeichen am Ende eines Funktionsnamens kennzeichnet Funktionen mit String-Ergebnis.

Beispiel:

```
OPENW 0
fl%=@fak_loop(15)
fr%=@fak_rekurs(10)
'
PRINT "Schleife:  fak(15) = ";fl%
PRINT "Rekursion: fak(10) = ";fr%
'
FUNCTION fak_loop(f%)
  w=1
  FOR j%=1 TO f%
    MUL w,j%
  NEXT j%
  RETURN w
ENDFUNC
'
FUNCTION fak_rekurs(f%)
  IF f%<2
    RETURN 1
  ELSE
    RETURN f%   *@fak_rekurs(PRED(f%))
  ENDIF
ENDFUNC
```

Die Fakultäten von 10 und 15 werden zum einen innerhalb einer Schleife und zum anderen rekursiv berechnet.

DEFFN func [(x1,x2,...)] = ausd
FN func [(y1,y2,...)]

func, x1, x2: var
ausd, y1, y2: exp

Der Befehl DEFFN erlaubt das Definieren von einzeiligen Funktionen. Die Definition dieser Funktion steht im Ausdruck „ausd", der sowohl einen numerischen als auch einen String-Ausdruck ergeben kann. Die Funktion wird nach ihrer Definition mit FN func oder dem Klammeraffen @ aufgerufen.

Die in der Funktionsdefinition enthaltenen Variablen werden als Parameter übergeben. In der Funktionsdefinition notiert man zu diesem Zweck die Variablennamen x1, x2,... die in ausd auftauchen. Beim Aufruf der Funktion können die Parameter y1, y2,... als numerischer oder String-Ausdruck übergeben werden.

Wenn die Variablen x1, x2,... auch global definiert sind, so können sie im Ausdruck ausd nicht angesprochen werden, da sie innerhalb von ausd die übergebenen Werte enthalten. Die Variablen x1, x2,... sind also für die Funktion lokale Variablen.

Die Anweisung FN kann auch durch @ ersetzt werden. Funktionen können beliebig verschachtelt werden, bei DEFFN allerdings nicht rekursiv (Abbruchbedingung nicht möglich).

Beispiele:

```
OPENW 0
DEFFN test(y,a$)=x–y+LEN(a$)
x=2
PRINT @test(4,"abcdef")
```

Es erscheint die Zahl 4 (2–4+6) im Fenster.

```
DEFFN first_last$(a$)=LEFT$(a$)+RIGHT$(a$)
b$=@first_last$("TEST")
PRINT b$
```

Es erscheint der Text „TT".

```
DEFFN hoch_vier(x)=x^4
DEFFN vierte_wurzel(x)=x^(1/4)
PRINT @vierte_wurzel(@hoch_vier(1024))
```

Es erscheint die Zahl 1024.

ON BREAK
ON BREAK CONT
ON BREAK GOSUB proc

proc: Name einer Prozedur

Diese drei Befehle legen die Reaktion auf das gleichzeitige Drücken der Tasten Control, Shift (nur linke Shift-Taste) und Alternate fest. Ein Drücken dieser Tastenkombination führt gewöhnlich zum Programmabbruch, kann aber auch zum Anspringen einer bestimmten Prozedur dienen. Zu diesem Zweck legt man mit ON BREAK GOSUB die anzuspringende Prozedur proc fest.

Mit ON BREAK CONT erreicht man, daß auf das Drücken der genannten Tastenkombination nicht mehr reagiert wird. ON BREAK schaltet die normale Reaktion (Programmabbruch) wieder ein.

Beispiel:

```
ON BREAK GOSUB test
PRINT "Drücken Sie CONTROL, SHIFT (links) und ALTERNATE"
DO
LOOP
'
PROCEDURE test
  PRINT "Das war's"
  ON BREAK
RETURN
```

Es erscheint die Aufforderung zum Drücken der Tastenkombination. Wird sie gedrückt, schaltet die Prozedur test die normale Break-Routine wieder ein.

ON ERROR
ON ERROR GOSUB proc
RESUME [NEXT]
RESUME [mar]

proc: Name einer Prozedur
mar: Name einer Marke

Das Auftreten eines Fehlers führt normalerweise zur Ausgabe einer Fehlermeldung und zum Programmabbruch. Mit ON ERROR GOSUB hat man die Möglichkeit, beim Auftreten eines Fehlers zur Prozedur proc zu verzweigen. In dieser Prozedur kann man die Reaktion auf Fehler festlegen. Mit dem Befehl ON ERROR schaltet man wieder auf die normale Fehlerbehandlung zurück, also auf Ausgabe einer Fehlermeldung und Programmabbruch.

Wenn ein Fehler auftritt, so wird automatisch ein ON ERROR-Befehl ausgeführt. Um auf mehrere hintereinanderfolgende Fehler reagieren zu können, muß in der Fehlerabfangroutine somit der Befehl ON ERROR GOSUB proc noch einmal enthalten sein.

Der RESUME-Befehl erlaubt eine spezielle Reaktionsweise auf aufgetretene Fehler. Er ist nur in der Fehlerbehandlungs-Prozedur sinnvoll. Mit RESUME NEXT kann erreicht werden, daß der nächste Befehl angesprungen wird, der hinter der Programmanweisung steht, die zum Fehler führte. RESUME mar führt dazu, daß die Marke mar angesprungen wird. Der Befehl RESUME ohne Angabe von NEXT oder einer Marke springt den Befehl, bei dem der Fehler auftrat, erneut an.

Beispiel:

```
ON ERROR GOSUB fehlerabfanger
ERROR 5
PRINT "und nochmal ..."
ERROR 5
PRINT "wird nicht erreicht"
'
PROCEDURE fehlerabfanger
  PRINT "Ok, Fehler abgefangen."
  RESUME NEXT
RETURN
```

Es erscheinen die Texte „Ok, Fehler abgefangen." und „und nochmal ..." auf dem Bildschirm, dann die durch ERROR 5 ausgelöste Fehlermeldung „Quadratwurzel nur für positive Zahlen". Die Marke für RESUME darf sowohl in einer Prozedur als auch im Hauptprogramm stehen.

ERROR x
ERR
ERR$(x)

x:aexp

Durch ERROR läßt sich der Fehler mit der Nummer x auslösen (Fehlermeldungstabelle siehe Anhang). Dies ist z.B. beim Austesten einer Fehlerbehandlungsroutine sinnvoll. In der Variable ERR steht die Nummer des aufgetretenen Fehlers. Mit ihrer Hilfe kann man in einer Fehlerbehandlungsroutine bestimmten Fehlern spezifische Reaktionen zuordnen.

Die Funktion ERR$ liefert den String der GFA-BASIC-Fehlermeldung mit der Nummer x.

Beispiele:

```
ON ERROR GOSUB fehlerabfangung
INPUT "Welchen Fehler hätten Sie denn gern: ",e
ERROR e
'
PROCEDURE fehlerabfangung
  PRINT "Dies war Fehler–Nr.: ";ERR
RETURN
```

Fragt den Benutzer nach der gewünschten Fehlernummer und gibt diese Nummer wieder aus.

EVERY ticks GOSUB proc
EVERY STOP
EVERY CONT
AFTER ticks GOSUB proc
AFTER STOP
AFTER CONT

ticks: iexp
proc: Name einer Prozedur

Mit Hilfe der Befehle EVERY und AFTER können Prozeduren nach Ablauf einer bestimmten Zeit ticks aufgerufen werden. Der Befehl EVERY führt dazu, daß die Prozedur proc alle ticks Zeiteinheiten aufgerufen wird; AFTER verursacht, daß diese Prozedur einmal nach Ablauf der ticks Zeiteinheiten bearbeitet wird.

Die Zeiteinheit wird dabei in zweihundertstel Sekunden angegeben (ticks=200 bedeutet also eine Sekunde). Die Zeitauflösung beträgt hierbei nur etwa ein Zehntel Sekunde, bei Mausbewegungen ist sie etwas höher.

Mit Hilfe von EVERY STOP kann das Anspringen der Prozedur nach Ablauf der Zeitfrist abgeschaltet werden, mit EVERY CONT wird es wieder fortgesetzt. Die Befehle AFTER STOP und AFTER CONT arbeiten analog.

Es wird jeweils nur nach vollständiger Abarbeitung eines Befehls geprüft, ob eine solche Prozedur ausgeführt werden soll, d.h. langsam ablaufende Befehle wie INPUT, QSORT, Dateioperationen oder ähnliches können diese Routinen behindern.

Beispiel:

```
OPENW 0
EVERY 4 GOSUB lines
linien!=TRUE
GRAPHMODE 3
DEFFILL 1,0
PLOT MOUSEX,MOUSEY
REPEAT
  IF MOUSEK=1
    EVERY STOP
  ELSE
    EVERY  CONT
  ENDIF
  DRAW TO MOUSEX,MOUSEY
UNTIL MOUSEK=2

PROCEDURE lines
  INC y%
  LINE 160,y%,320,y%
  IF y%=200
    y%=0
  ENDIF
RETURN
```

Läßt Linien in der rechten Fensterhälfte von oben nach unten wandern und erlaubt gleichzeitig das Zeichnen mit der Maus. Das Drücken der linken Maustaste schaltet die wandernden Linien ein bzw. aus. Das Programm kann durch Drücken der rechten Maustaste abgebrochen werden.

```
PRINT "In 3 Sekunden folgt ein Text,"
PRINT "wenn Sie keine Taste drücken."
AFTER 600 GOSUB text
REPEAT
UNTIL INKEY$<>"" OR RAUS!
AFTER STOP
'
PROCEDURE text
  PRINT
  PRINT "Hier ist der Text"
  raus!=TRUE
RETURN
```

Wird in den drei Sekunden nach dem Programmstart keine Taste gedrückt, so erscheint ein Text. Wird eine Taste gedrückt, so beendet dies das Programm.

REM x
' x
< Befehlszeile> ! x

x: beliebiger Text

In einer Zeile, die mit REM oder ' beginnt, können beliebige Texte stehen. Diese Texte unterliegen nicht der Syntaxkontrolle des Editors und werden bei der Programmausführung nicht beachtet. Anstatt des Apostroph können auch ähnliche Zeichen benutzt werden, die dann bei der Eingabe automatisch in ein Apostroph umgewandelt werden. Der Vorteil liegt darin., daß diese alternativen Zeichen über die Tastatur eventuell besser erreichbar sind.

Außerdem kann an das Ende einer Befehlszeile hinter dem Ausrufezeichen ! ein beliebiger Kommentar angefügt werden. Im Zusammenhang mit DATA-Anweisungen ist die Benutzung dieses Zeichens allerdings nicht möglich. Auch bei INLINE ist kein Kommentar möglich.

Statt des Apostrophs (ALT-„A") können auch die Einzelakzente (^) oder das Ausrufezeichen (!) verwendet werden.

Beispiel:

```
REM Kommentar
' PRINT "Kommentar"
PRINT "REM"  ! Kommentar
```

Auf dem Bildschirm erscheint das Wort REM.

GOTO mar

mar: vom Programmierer definierte Marke

Mit Hilfe einer Marke mar: kann man Stellen im Programm festlegen, die mit dem Befehl GOTO angesprungen werden können. Die Programmausführung wird dann an der Position der Marke fortgesetzt. Die Marke kann aus Buchstaben, Ziffern, Unterstrichen und Punkten bestehen. Sie darf auch, im Unterschied zu Variablennamen, mit einer Ziffer beginnen, muß aber mit einem Doppelpunkt enden. Beim Ansprung der Marke mit GOTO wird der Doppelpunkt allerdings nicht angegeben.

Es ist mit Hilfe des GOTO-Befehls nicht möglich, in Prozeduren oder Funktionen hinein- oder aus ihnen herauszuspringen. Dies gilt auch für FOR NEXT-Schleifen. GOTO-Befehle machen Programme meist sehr schnell unübersichtlich, weshalb man sie nach Möglichkeit vermeiden sollte.

Beispiel:

```
OPENW 1
PRINT "Stelle 1"
GOTO sprungmarke
PRINT "Stelle 2"
sprungmarke:
PRINT "Stelle 3"
```

Im Fenster erscheinen die Texte „Stelle 1“ und „Stelle 3“.

PAUSE x
DELAY x

x: aexp

Der Befehl PAUSE hält die Programmausführung für x / 50 Sekunden an. DELAY hat eine ähnliche Wirkung, jedoch wird die Ausführung des Programms für x Sekunden unterbrochen.

Da „x" auch eine beliebige Fließkommazahl sein kann, sollten Sie in diesem Fall wissen, daß die maximale Auflösung des Timers nur 1/50 Sekunde beträgt. Kleinere Werte haben also immer die selbe Wirkung: es wird 1/50 Sekunde gewartet. Darüberhinaus müssen Sie noch beachten, daß beim Amiga aufgrund des Multitasking-Betriebssystems, Verzögerungen entstehen können. Delay wartet also mindestens eine Sekunde.

Beispiel:

```
PRINT "Start"
PAUSE 100
PRINT "eine Pause"
DELAY 2
PRINT "Ende"
```

Es erscheinen der Text „Start" und zwei Sekunden später „eine Pause", nach weiteren 2 Sekunden „Ende".

END
EDIT
STOP

Diese Befehle dienen der Beendigung des Programms. Der Befehl END hält die Programmausführung an und läßt einen Alert-Requester mit dem Text „Programmende" erscheinen. Nach dem Anwählen der einzigen Wahlalternative kehrt GFA-BASIC 3.0 in den Editor zurück.

EDIT beendet die Programmausführung und kehrt sofort in den Editor zurück. STOP läßt einen Alert-Requester mit den Wahlalternativen STOP und CONT erscheinen. Durch die Wahl von CONT kann man die Programmausführung fortsetzen. Nach der Wahl von STOP geht GFA-BASIC in den Direktmodus. Dort kann man nun die Werte von Variablen verändern oder abfragen und durch CONT die Programmausführung wieder fortsetzen.

Möchten Sie das Programm beenden, ohne in den Editor zu gelangen, müßen Sie QUIT oder SYSTEM benutzten.

Beispiel:

```
x=3
STOP
PRINT x
```

Wählen Sie den Button „STOP", wenn der entsprechende Alert-Requester erscheint. Geben Sie nun im Direktmodus folgende Befehle ein:

```
PRINT x
```

Es erscheint die Zahl 3. Geben Sie ein:

```
x=4
CONT
```

Der letzte Befehl des Listings (PRINT x) wird nun abgearbeitet. Es erscheint die Zahl 4, also der im Direktmodus angegebene Wert.

NEW

Diese Anweisung löscht ein Programm. Im Direktmodus erfolgt eine Sicherheitsabfrage. Im Editor ist dieser Befehl durch Shift-F4 oder Mausklick (mit Sicherheitsabfrage) zu erreichen.

Dieser Befehl läßt sich auch durchaus in einem Programm einsetzten. So könnten Sie beispielsweise ein Programm schreiben, daß eine Initialisierung ihrer Standard-Arbeitsumgebung vornimmt. Am Ende des Programms bewirkt der Befehl NEW, daß diese Initialisierungsprogramm gelöscht wird.

LOAD f$

f$: sexp

Der Befehl LOAD dient zum Laden eines GFA-BASIC-Programms. Der String-Ausdruck f$ enthält den Zugriffspfad der gewünschten Datei. Ohne Angabe einer Extension wird voreingestellt .GFA benutzt.

Beispiel:

```
LOAD "DF0:TEST.GFA"
```

Lädt die Programmdatei TEST.GFA aus dem Wurzelverzeichnis von Laufwerk DF0.

SAVE f$
PSAVE f$

f$: sexp

Der Befehl SAVE speichert eine Programmdatei unter dem in f$ angegebenen Namen ab. Bei Verwendung des Befehls PSAVE wird die angegebene Datei mit List-Schutz abgespeichert (d.h., die Datei kann nach erneutem Laden mit LOAD nicht gelistet werden, sondern wird sofort ausgeführt). Ohne Angabe einer Extension wird .GFA verwendet.

Beispiel:

```
SAVE "DF0:TEST.GFA"
```

Speichert die aktuelle Programmdatei unter dem Namen TEST.GFA auf Laufwerk DF0 ab.

LIST [f$]
LLIST [f$]

f$: sexp

Der Befehl LIST zeigt das aktuelle Programm auf dem Editor-Screen an. Optional kann ein Zugriffspfad angegeben werden, unter dem die Programmdatei im ASCII-Format abgespeichert werden kann. Programmdateien, die mit MERGE in andere Programme eingefügt werden sollen, müssen mit LIST oder SAVE,A (aus der Editor-Menüleiste) im ASCII-Format gespeichert sein.

Ohne Angabe einer Extension wird voreingestellt .LST verwendet. Mit LLIST kann das aktuelle Programm auf den Drucker ausgegeben werden.

Das Drucker-Listing kann nur durch Ausschalten des Druckers unterbrochen werden. Danach dauert es noch ca. 30 Sekunden, bis das Programm weiter ausgeführt bzw. in den Editor zurückkehrt. (vgl. LLIST und die Punktbefehle im Abschnitt über den Editor). Es ist aber auch möglich, die Ausgabe umzulenken. (vgl. OPEN)

Beispiel:

```
LIST "DF1:TEST.LST"
```

Speichert das aktuelle Programm unter dem Namen „TEST.LST“ im ASCII–Format auf Laufwerk DF1 ab.

```
.ll 70
.pl 66
LLIST
```

Gibt das aktuelle Programm mit einer Zeilenlänge von 70 Zeichen und 66 Zeilen pro Seite auf den Drucker aus.

CHAIN f$

f$: sexp

Der Befehl CHAIN ermöglicht das Laden eines GFA-BASIC-Programms in den Arbeitsspeicher und startet das Programm. Ohne Angabe einer Extension wird voreingestellt .GFA verwendet.

Beispiel:

```
CHAIN "DF0:ZUSATZ.GFA"
```

Lädt die Programmdatei ZUSATZ.GFA und startet das Programm.

RUN [f$]

f$: sexp

Der Befehl RUN startet das aktuelle Programm. Wird zusätzlich ein vollständiger Dateiname angegeben, so wird das entsprechende Programm geladen und gestartet.

Beispiel:

```
RUN "DF0:TEIL_2.GFA"
```

Lädt und startet das Programm mit dem Namen TEIL_2.GFA von Laufwerk DF0.

SYSTEM [n]
QUIT [n]

n: iexp

Die Befehle SYSTEM und QUIT sind in der Wirkung gleichwertig. Sie beenden den Programmlauf und verlassen GFA-BASIC. Wenn der Interpreter verlassen wurde, wird an das aufrufende Programm (im Normalfall das CLI) eine Rückgabevariable übergeben. Dabei gilt die Konvention, daß Null einen fehlerfreien Durchlauf signalisiert.

TRON
TRON #n
TROFF

n: iexp

Der Befehl TRON (trace on) führt dazu, daß jeder ausgeführte Befehl auf dem Editor-Screen aufgelistet wird. Diese Liste kann unter Angabe einer Kanalnummer in eine Datei, auf den Drucker oder auf die serielle Schnittstelle usw. umgeleitet werden. Der Befehl TROFF schaltet dies wieder ab.

Beispiel:

```
PRINT "Start:"
TRON
FOR i%=1 TO 5
  PRINT i%
NEXT i%
TROFF
PRINT "Ende."
```

Es erscheint das Wort „Start:". Dann die Zahlen von 1 bis 5 und die Befehle, die zu deren Darstellung führen. Danach das Wort „Ende".

```
OPENW 0
OPEN "O",#1,"tron.lst"
TRON #1
FOR i%=1 TO 10
  PRINT i%
NEXT i%
TROFF
CLOSE #1
'
OPEN "O",#2,"prn:"
TRON #2
FOR i%=10 TO 320 STEP 10
  LINE i%,0,i%,100
NEXT i%
TROFF
CLOSE #2
```

Es erscheinen die Zahlen von 1 bis 10 sowie die dazu notwendigen Befehle und eine Reihe von vertikalen Linien im Abstand von 10 Pixeln. Diese Befehlsausgabe erfolgt auf Diskette bzw. Drucker.

TRON proc
TRACE$

proc: Name einer Prozedur

Mit Hilfe des Befehls TRON proc kann eine Prozedur angegeben werden, die vor der Bearbeitung eines jeden Befehls aufgerufen wird. Die Variable TRACE$ enthält den Befehl, der als nächster abgearbeitet wird. Die Anweisung TRON proc ermöglicht in Zusammenarbeit mit TRACE$ eine sehr effiziente Fehlersuche. So kann z.B. in Abhängigkeit vom nächsten zu verarbeitenden Befehl eine Ausgabe von bestimmten Variablen auf dem Drucker erfolgen, so daß die Veränderungen einer Variablen während des Programmlaufs verfolgt werden können.

Wichtig ist, daß die TRON-Prozedur möglichst wenig stören sollte. Also keine PRINT-Befehle in Bildschirmmasken (TEXT).

Beispiel:

```
OPENW 0
TRON tr_proc
GRAPHMODE 3
DO UNTIL MOUSEK
  x1%=100+RAND(200)
  y1%=100+RAND(100)
  x2%=200+RAND(200)
  y2%=200+RAND(100)
  PBOX x1%,y1%,x2%,y2%
LOOP
'
PROCEDURE tr_proc
    PRINT AT(1,5);SPACE$(80);
    PRINT AT(1,5);LEFT$(TRACE$,79);
    PAUSE 20
  ENDIF
RETURN
```

Dieses Programm zeichnet zufällig verteilte Rechtecke auf den Bildschirm. Das Drücken der Control-Taste läßt die gerade abgearbeiteten Befehle auf dem Bildschirm erscheinen. Durch Drücken einer Maustaste wird das Programm beendet.

$ text

text: Folge von beliebigen Zeichen

Der Befehl $, der vom Interpreter wie ein REM behandelt wird, dient zur Steuerung des Compilers. Eine genaue Beschreibung finden Sie in der Anleitung der Compiler-Version zum GFA-BASIC 3.0.

INLINE adr,anz

adr: 4-Byte-Integervariable, keine Feldvariable
anz: Integerkonstante, die kleiner als 32700 ist

Hinter diesem Befehl ist kein Kommentar möglich, da intern genau an der Stelle, wo sonst der Kommentar steht, der gewünschte Speicherplatz reserviert wird. Anfänglich wird der Speicherplatz gelöscht (d.h. mit Null-Bytes gefüllt). Dieser Speicherbereich beginnt immer an einer geraden Adresse. Beim Ausführen von INLINE wird diese Adresse in die Integervariable adr geschrieben. Beim Speichern oder Laden des Programms wird der reservierte Speicherbereich mitgespeichert bzw. mitgeladen.

Positioniert man den Cursor auf die Programmzeile, die den INLINE-Befehl enthält und drückt die Help-Taste, dann erscheint in der obersten Zeile des Editors eine Menüzeile mit den Einträgen LOAD, SAVE, CLEAR und DUMP. Mit den Menüpunkten „LOAD" und „SAVE" ist es möglich, Dateien in die Zeile mit dem INLINE-Befehl zu laden oder abzuspeichern. Dabei ist die Extension .INL voreingestellt. Der Menüpunkt „CLEAR" dient zum Löschen des reservierten Speicherbereiches. In diesen Speicherbereich können z.B. Bilder, Tabellen oder Assembler-Programme geladen werden. Mit DUMP erfolgt ein HEX-Dump auf den Drucker.

C:adr([x,y,...])

adr:avar (mindestens 32 Bit, besser adr%)
x, y: iexp

Die Funktion C ruft ein in C oder Assembler geschriebenes Unterprogramm auf, das an der Adresse „adr" steht. In den Klammern können die Parameter „x, y,..." übergeben werden. Die Parameterübergabe erfolgt wie in C. Die Parameter können als 32-Bit-Langwort oder 16-Bit-Wort übergeben werden, wobei die Übergabe als 16-Bit-Wert voreingestellt ist. Langworte können übergeben werden, indem man vor dem entsprechenden Parameter ein „L:" setzt. Beim Aufruf dieser Funktion werden auf dem Stack die Rücksprungadresse und dann die Parameter abgelegt. So führt z.B.:

VOID C:adr%(L:x,W:y,z)

auf dem Stack zu folgender Situation:

(sp) → Rücksprungadresse
4(sp) → x (4 Bytes)
8(sp) → y (2 Bytes)
10(sp) → z (2 Bytes)

Der von der Funktion zurückgemeldete Wert ist der Inhalt des Registers d0 beim Rücksprung aus der Unterroutine (der mit RTS erfolgen muß).

Beispiel:

Das hier verwendete Assembler-Programm füllt einen Speicherbereich (z.B. ein Integer-Array) ab einer bestimmten Adresse mit den Zahlen (Langworte) von 0 bis n.

```
206F0004        move.l    4(sp),a0        ;Anfangsadresse
202F0008        move.l    8(sp),d0        ;Anzahl der Werte
7200            moveq.l   #d0,d1          ;Zähler
6004            bra.s     ct_2            ;Einsprung in Schleife
20C1      ct_1:move.l     d1,(a0)+        ;Schleife, Wert schreiben
5281            addq.l    #d1,d1          ;Zähler erhöhen
```

```
B081    ct_2:cmp.l   d1,d0     ;fertig ?
64F8       bcc.s     ct_1     ;nein, weiter —>
4E75       rts                ;Rücksprung ins GFA–BASIC
```

Das BASIC-Programm ist:

```
FOR i%=1 TO 11
  READ a%
  asm$=asm$+MKI$(a%)
NEXT i%
DATA $206F,$0004,$202F,$0008,$7200,$6004,$20C1,$5281,$B081,$64F8,$4E75
'
DIM x%(10000)
asm%=V:asm$
~C:asm%(L:V:x%(0),L:10000)
PRINT "z.B. x%(12): ";x%(12)
```

Füllt das Array x%() mit den Zahlen von 0 bis 10000. Bei dieser Anwendung entspricht die Ausführung der C:-Funktion den Befehlen:

```
FOR i%=1 TO n%
  x%(i%)=i%
NEXT i%
```

Eine andere Möglichkeit, ein Assembler-Programm in ein GFA-BASIC 3.0-Programm einzufügen, bietet der INLINE-Befehl. Zuerst ein File erzeugen, in dem das oben abgedruckte Assembler-Programm enthalten ist, z.B. (nach der oben abgedruckten READ-DATA-Schleife) durch:

```
BSAVE "COUNT.INL",V:asm$,22
```

Nun geben Sie das folgende Programm ein:

```
INLINE asm%,22
DIM x%(10000)
~C:asm%(L:V:x%(0),L:10000)
```

Dann in der Zeile, in der INLINE steht, die Help-Taste drücken, wählen Sie LOAD an und laden COUNT.INL ein. Nun kann man dieses Programm mit SAVE abspeichern.

MONITOR [x]

x: iexp

Dieser Befehl dient dem Aufruf von Assembler-Unterroutinen, von Debug-Programmen oder anderen Hilfsprogrammen. Zu diesem Zweck muß der Illegal Instruktion Vektor (Adresse 16) auf die Adresse der Unterroutinen „gebogen" werden. Der MONITOR-Befehl erzeugt dann eine Illegal Instruction Exception, die zum Ansprung des Unterprogramms führt, das mit RTE (Return from Exception) enden muß. Der Parameter x ist ein Übergabewert, der ins Register d0 geschrieben wird. Ein Anwendungsbeispiel ist in einem Programm wie dem unter C: beschriebenen, einen Debugger zu verwenden. Dazu wird GFA-BASIC vom Debugger aus geladen und gestartet. Nach Eingabe des Programms, direkt hinter den INLINE-Befehl fügen Sie ein: MONITOR asm%. Nach Start des Programms wird sich der Debugger wegen „Illegal Instruction" melden. Nun kann man sich den Bereich, auf den d0 zeigt, disassemblieren lassen oder editieren. Anschließend kann man, eventuell nach Drücken von Shift-Alternate-Control, die Ausführung des BASIC-Programms mit dem GO-Befehl des Debuggers fortsetzen.

VORSICHT:
Bei Benutzung dieses Befehls müßen Sie vorher sicherstellen, daß auch ein Maschinensprache-Monitor o.ä. den MONITOR-Vektor initialisiert hat. GFA-BASIC hat keine Möglichkeit vor dem Sprung zu dieser Vektor-Adresse festzustellen, ob wirklich ein Monitor vorhanden ist.. Ist er nicht vorhanden, ist ein Absturz des Programms oder des Systems die Regel (Guru-Meditation 4).

CALL adr([x,y,...])

adr: avar (mindestens 32 Bit, besser adr%)
x, y: iexp

Mit Hilfe des Befehls CALL können ebenfalls Assembler- oder C-Unterprogramme aufgerufen werden. Dabei ist „adr" die Adresse, ab der das Assembler-Programm im Speicher liegt. Es besteht die Möglichkeit, eine Parameterliste zu übergeben. Nach dem Aufruf von CALL befindet sich die Rücksprungadresse auf dem Stack (die Assembler-Routine muß mit RTS enden). Dahinter folgt die Anzahl der übergebenen Parameter als 16-Bit-Wert und schließlich die Adresse, ab der die Parameter im Speicher liegen, als 32-Bit-Wert. Alle Parameter werden als Langworte (32 Bit) interpretiert.

Es besteht hier die Möglichkeit, auch Strings als Parameter zu übergeben. In diesem Fall ist der Übergabewert die Anfangsadresse des Strings.

Stackaufbau

(sp) → Rücksprungadresse
4(sp) → Anzahl der Parameter (16 Bit)
6(sp) → Adresse des Parameterfeldes (32 Bit)

RCALL adr,reg%()

reg%(): Name 4-Byte-Integer-Arrays
adr:iexp

Der Befehl RCALL ermöglicht es, die Register vor dem Start der Assembler-Routine mit Werten vorzubelegen und die Registerinhalte nach Ablauf der Routine abzufragen. Diesem Zweck dient das Array reg%(), dessen Elemente vom 4-Byte-Integer-Typ sein müssen und das mindestens 16 Elemente haben muß. Vor dem Start der Assembler-Routine werden die Einträge dieses Arrays in die Register kopiert und nach Ablauf der Routine die Inhalte der Register in die entsprechenden Array-Elemente geschrieben. Dabei gilt folgende Zuordnung (bei OPTION BASE 0):

Datenregister	d0 bis d7	in reg%(0)	bis	reg%(7)
Adreßregister	a0 bis a6	in reg%(8)	bis	reg%(14)
User-Stack-Pointer	(a7)	in reg%(15)	(nur Rückgabe)	

Beispiel:

```
DIM r%(15)
r%(14)=_IntBase
r%(8)=0
RCALL _IntBase-96,r%()
```

Dieses Programm ruft die Betriebssystemroutine DisplayBeep auf und entspricht ~DisplayBeep(0).

8 – Grafik

SETCOLOR

SETCOLOR register,rot,grün,blau
SETCOLOR register,mischwert
COLOR front,back,outl

register,rot,gruen,blau,mischwert,front,back,outl: iexp

Die erste Variante von SETCOLOR bestimmt den Anteil der Farben Rot, Grün und Blau in einem bestimmten Farbregister. Dabei wird die Intensität der Farbanteile durch eine Skala von 0 (gering) bis 15 (groß) festgelegt. Die Anzahl der verfügbaren Farben ist abhängig von der aktuellen Screen-Auflösung. In der zweiten Variante wird die Farbeinstellung über folgende Formel berechnet: mischwert = rot * 512 + grün * 16 + blau, wobei die Werte für rot, gruen und blau wieder aus der Skala von 0 bis 15 stammen. Hierfür eignet sich besonders die hexadezimale Darstellung (Setcolor 1,&hF00 (=hellrot)).

Da für jeden geöffneten Screen eine eigene Farbpalette gesetzt werden kann, gelten diese beiden Befehle jeweils für den aktuellen Screen. Der Befehl COLOR bestimmt die Zeichenfarbe. Dabei steht in den Ausdrücken front,back und out ein von der jeweiligen Screenauflösung abhängiger Wert zwischen 0 und 63.

In front wird dabei die Zeichenfarbe für den Vordergrund und in back die für den Hintergrund angegeben. Die Hintergrundfarbe wird nur im Zeichen modus GRAPHMODE 1 (JAM2) dargestellt. Der Ausdruck out enthält den Farbwert für die Umrahmung bei Füllbefehlen (PBOX, PCIRCLE, PELLIPSE, POLYFILL).

Beispiel:

```
OPENS 1,0,0,640,200,2,32768
SETCOLOR 0,0
```

Öffnet einen Screen und setzt die Farbe Schwarz in das Farbregister 0.

DEFFILL [farbe] , [stil] , [muster]
DEFFILL [farbe] , bitmuster$

farbe, stil, muster: iexp
bitmuster$: sexp

Dieser Befehl legt das Füllmuster für die Befehle PBOX, PCIRCLE, PELLIPSE, POLYFILL und FILL fest. Er bestimmt Farbe, Stil und Muster der Füllung und ermöglicht es, selbsterstellte Muster zu definieren. Für die Variable farbe können, je nach Auflösung, Werte von 0 bis 63 verwendet werden (siehe Übersicht zu Beginn des Kapitels Screens). Für „stil" gilt folgende Zuordnung:

0 = Leer
1 = Ausgefüllt
2 = Punktiert
3 = Schraffiert

Mit „muster" kann aus 24 Punkt- oder 12 Linienmustern eines ausgewählt werden (siehe Anhang: Füllmustertabelle).

In der zweiten Befehlsvariation wird in bitmuster$ die Bit-Information zur Definition des Füllmusters übergeben. Diese Informationen müssen im Wort-Format vorliegen, wozu sich der Befehl MKI$ anbietet.

Durch die Stringlänge wird automatisch die Höhe des Füllmusters festgelegt. Dieses Höhe ist immer eine Zweier-Potenz (1,2,4,8,...), die Stringlänge jeweils doppelt so groß (2,4,8,16,...), da die Füllmuster 16 Bit breit sind. Für mehrfarbige Füllmuster ist ein Dummy-Bereich an den String anzuhängen, dann ist das Füllmuster ebenenweise nacheinander anzugeben (bei 4 Bitplanes und 16 Pixeln Höhe also 4*(16*2)=128 Zeichen +1). Die maximale Stringlänge ist 257.

Vorne stehende Parameter können weggelassen werden, wenn man die parametertrennenden Kommata aufführt. So wählt z.B. DEFFILL ,2,4 das Füllmuster 2,4 aus und läßt die Füllfarbe unverändert.

Beispiele:

```
OPENW 0
GRAPHMODE 1
DO
  COLOR RAND(64),RAND(64),RAND(64)
  DEFFILL 1,RAND(2)+2, RAND(24)+1
  PBOX RAND(320),RAND(256),RAND(320),RAND(256)
LOOP
```

Zeichnet verschieden große mit zufällig ausgewählten Füllmustern gefüllte Rechtecke in das Fenster.

```
FOR i=1 TO 16
  f$=f$+MKI$(RAND(65535))
NEXT i
DEFFILL 1,f$
PBOX 100,100,150,150
```

Zeichnet eine Box mit zufällig generiertem Füllmuster.

BOUNDARY n

n: iexp

Die Anweisung BOUNDARY schaltet die automatische Umrahmung der Füllfläche ein oder aus. Wenn n gleich Null ist, wird die Umrahmung ausgeschaltet und für n ungleich Null eingeschaltet.

Beispiel:

```
DEFFILL 1,2,2
BOUNDARY 1              ! Umrahmung einschalten
PBOX 50,50,100,100
BOUNDARY 0              ! Umrahmung ausschalten
PBOX 150,50,200,100
```

Zeichnet ein gefülltes Rechteck mit und eines ohne Umrahmung.

DEFLINE def

def: iexp

Der Befehl DEFLINE legt das Aussehen der Linien fest, die mit den Befehlen LINE, BOX, CIRCLE, ELLIPSE und POLYLINE gezeichnet werden.

Im Ausdruck „def" wird ein 16 Bit-Werte festgelegt, der das Aussehen der Linie bestimmt. Ein gesetztes Bit in der Definition entspricht einem gesetzten Punkt. Im Zeichenmodus GRAPHMODE 1 werden die in der Definition nicht gesetzten Bits, in der Hintergrundfarbe dargestellt.

Wird für „def" ein Wert zwischen 0 und 6 angegeben werden folgende Standard-Bitmuster benutzt:

0 Linie in Hintergrundfarbe

1 Durchgezogene Linie

2 Gestrichelte Linie mit kleinen Abständen

3 Gepunktete Linie

4 Strichpunktierte Linie

5 Gestrichelte Linie mit großen Abständen.

6 Das Linienmuster wird: Strich - Punkt - Punkt ...

Beispiel:

```
OPENW 0
DEFLINE &X1111000011110000
COLOR 1,2
GRAPHMODE 1
LINE 20,20,200,100
```

Zeichnet eine zweifarbige Linie in das Fenster.

GRAPHMODE n

n: iexp

Der Befehl GRAPHMODE legt fest, in welcher Weise Grafikausgaben auf dem Bildschirm miteinander verknüpft werden. Dabei steht der numerische Ausdruck n für 4 mögliche Modi:

0	JAM1	transparent (durchsichtig)
1	JAM2	replace (ersetzen)
2	COMPLEMENT	xor (invertieren)
4	INVERSVID	reverse transparent (invertiert und durchsichtig)

Hat n den Wert 0 (dies ist der voreingestellte Modus), werden altes und neues Muster mit OR verknüpft. Für n=1 überdeckt das neue Muster das früher gezeichnete. Bei n=2 wird jedes Pixel gesetzt, welches vorher nicht gesetzt war und jedes Pixel gelöscht, das vorher gesetzt war (die Muster werden also invertiert XOR-Verknüpfung).

Im Modus 4 wird das neue Muster zuerst invertiert und dann mit dem alten Muster OR verknüpft. Es sind auch Kombinationen möglich, also 6=INVERSVID+COMPLEMEND

Beispiel:

```
OPENW 0
COLOR 1,0,1
FOR i%=1 TO 4
  GRAPHMODE i%-1
  DEFFILL 1,2,18+i%
  PBOX (75 *i%)-50,20,75*i%,70
  DEFFILL 1,3,i%
  PBOX (75 *i%)-70,50,75*i%-20,100
NEXT i%
```

Das Beispiel zeigt 4 Rechteckpaare mit unterschiedlichen Mustern, die miteinander in den 4 Modi verknüpft sind.

RASTPORT adr

adr:iexp

Rastports sind die Zeichenebenen von Screens, Fenstern und Requestern. Mit dem Befehl RASTPORT kann der aktuelle Ausgaberastport festgelegt werden. Ist die Rastportadresse „adr" gesetzt, erfolgen alle Grafikausgaben auf diesem Rastport, bis entweder ein neuer Rastport gesetzt oder in „adr" der Wert 0 angegeben wird.

Wird ein neuer Screen geöffnet, kann die Adresse der Screen-Struktur mit der Funktion SCREEN(index) erfragt werden. Nun kann man die Rastportadresse des Screens auslesen.

Beachten Sie bitte, daß bei der Ausgabe auf Screen-Rastports keinerlei Clipping erfolgt — der Programmierer ist dafür selbst verantwortlich.

Beispiel:

```
RASTPORT SCREEN(1)+84
OPENW 2
RASTPORT {WINDOW(2)+50}
```

Zuerst wird der Rastport des geöffneten Screens als aktueller Rastport gesetzt, danach der des Windows. Beachten Sie bitte die Offsets 84 bzw. 50, die die Position des Rastport-Zeigers in der Screen bzw. Window-Struktur festlegen.

CLIP x,y,w,h [OFFSET x0,y0]
CLIP x1,y1 TO x2,y2 [OFFSET x0,y0]
CLIP #n [OFFSET x0,y0]
CLIP OFFSET x,y
CLIP OFF

x, y, w, h, x0, y0, x1, y1, x2, y2, n: iexp

Diese Gruppe von Anweisungen dient dem „Clipping". Darunter versteht man die Begrenzung von Grafikausgaben auf einen beliebigen, rechteckigen Fensterausschnitt. Zur Festlegung dieses Fensterausschnittes (clipping rectangle) müssen die Koordinaten der diagonal gegenüberliegenden Eckpunkte bzw. Breite und Höhe des Begrenzungsrechtecks übergeben werden.

Dabei werden verschiedene Varianten des CLIP-Befehls unterschieden. CLIP x,y,w,h ermöglicht die Angabe der linken x-Koordinate „x" und der oberen y-Koordinate „y" sowie Breite „w" und Höhe „h" des Begrenzungsrechtecks.

Eine weitere Möglichkeit bietet der Befehl CLIP x1,y1 TO x2,y2. Hierbei werden die Koordianten der diagonal gegenüberliegenden Eckpunkte (x1,y1) und (x2,y2) übergeben.

Die dritte Variante ermöglicht die Begrenzung der Ausgaben auf die Innenkanten des Fensters mit dem Index „n". Bei den genannten Varianten ist es mit dem optionalen Befehlszusatz OFFSET x0,y0 möglich, den Ursprung für die Grafikausgaben auf den Punkt mit den Koordianten x0,y0 festzulegen. Weiterhin kann der Befehl OFFSET x0,y0 auch alleine stehen, wobei der gleiche Zweck erfüllt wird und der Ursprung für die Grafikausgaben ebenso auf den Punkt (x0,y0) gelegt wird. Das „clipping" wird mit Hilfe des Befehls CLIP OFF abgeschaltet. Die Begrenzung der Grafikausgaben gilt nicht für die Befehle GET, PUT.

PLOT / LINE / DRAW

PLOT x,y
LINE x1,y1,x2,y2
DRAW [TO] [x,y]
DRAW [x1,y1] [TO x2,y2] [TO x3,y3] [TO ...]

x, y, x1, y1, x2, y2: iexp

PLOT zeichnet einen Punkt mit den Koordinaten x,y auf den Bildschirm. LINE zeichnet eine Linie zwischen den Koordinatenpaaren x1,y1 und x2,y2. Stil und Farbe dieser Linie können mit DEFLINE und COLOR festgelegt werden.

DRAW x,y entspricht dem Befehl PLOT. Mit Hilfe von DRAW TO x,y wird eine Linie zwischen den Koordinaten x,y und dem letzten gesetzten Punkt gezeichnet. Dabei ist es gleichgültig, ob dieser Punkt durch PLOT, LINE oder DRAW gesetzt wurde. Eine weitere Variante des DRAW-Befehls stellt DRAW x1,y1 TO x2,y2 dar. Dies entspricht dem LINE-Befehl. Zusätzlich können weitere Koordinatenangaben angehängt werden. Damit können z.B. Vielecke erzeugt werden. Schließlich ermöglicht die letzte Variante des DRAW-Befehls die Angabe von Kommandos, die Ähnlichkeit mit bestimmten Grafik-Kommandos von LOGO (Turtle-Grafik) und der Standard-Plottersprache HPGL von Hewlett-Packard haben. Damit ist es möglich, einen Plotter auf dem Bildschirm zu simulieren.

Beispiele:

```
OPENW 0
x=50
y=50
farbe=POINT(x,2 *y)
PRINT farbe
PLOT x,2 *50
LINE 10,10,200,100
PRINT POINT(x,100)
```

Zeichnet einen Punkt und ermittelt dessen Zeichenfarbe.

```
DO
  MOUSE mx,my,mk
  IF mk=1
    DRAW TO mx,my
  ENDIF
  EXIT IF mk=2
LOOP
```

Bei gedrückter linker Maustaste wird eine Verbindung zwischen den aktuellen Mauskoordinaten mx,my und dem letzten gesetzten Punkt gezeichnet. Die Schleife wird durch Drücken der rechten Maustaste verlassen.

DRAW ausdruck
DRAW(i)
SETDRAW

i: iexp
ausdruck: eine Folge von sexp und aexp, der erste muß sexp sein, Trennung durch Komma, Semikolon oder Hochkomma

Mit DRAW wird ein imaginärer Stift über den Bildschirm bewegt, der relativ zeichnen kann. Dies ähnelt sehr den Grafikbefehlen der Programmiersprache Logo. Die Parameter der einzelnen Befehle sind jeweils Fließkommazahlen, die auch in Strings angegeben werden können. Es gibt die Kommandos:

FD n (ForwarD)	Bewegt den Stift n Pixel vorwärts.
BK n (BacKward)	Bewegt den Stift n Pixel rückwärts.
SX x (Scale X)	Skaliert die „Stiftbewegung" für FD und BK
SY y (Scale Y)	Skaliert mit dem angegebenen Faktor. Die Skalierung mit SX und SY wirkt nur auf die Befehle FD und BK. Mit SX0 bzw. SY0 wird die Skalierung ausgeschaltet (schneller als die Skalierung mit dem Faktor 1 (SX1, SY1)).
LT w (Left Turn)	Gibt den Winkel „w" an, um den die Zeichenrichtung nach links gedreht wird.
RT w (Rigth Turn)	Analog nach rechts.
TT w (Turn To)	Gibt den absoluten Winkel „w" an, dazu gilt folgende Zuordnung:

0°
↑
270° ← Nullpunkt → 90°
↓
180°

Die Angaben für w=Winkel erfolgen in Grad.

MA x,y (Move Absolute)	Bewegt den Stift an die absoluten Koordinaten x und y.
DA x,y (Draw Absolute)	Bewegt den Stift an die absoluten Koordinaten x und y und zeichnet eine Linie in der aktuellen Farbe von der letzten Position zum Punkt (x,y).

MR xr,yr (Move Relative)	Wie „MA“, jedoch in Bezug zur letzten Position.
DR xr,yr (Draw Relative)	Wie „DR“, jedoch in Bezug zur letzten Position. Eine Verkürzung für den Ausdruck DRAW "MA",x,y,"TT",w stellt der Befehl SETDRAW x,y,w dar.
CO c (Color)	Setzt die Farbe „c“ als Zeichenfarbe (vgl. auch Parameter bei COLOR-Befehl).
PU (Pen Up)	Hebt den Stift.
PD (Pen Down)	Senkt den Stift.

Zusätzlich stehen die folgenden Nachfragefunktionen zur Verfügung:

DRAW(0)	liefert x-Position (als Fließkommawert)
DRAW(1)	liefert y-Position (als Fließkommawert)
DRAW(2)	liefert Winkel in Grad (als Fließkommawert)
DRAW(3)	liefert Skalierung der x-Achse (als Fließkommawert)
DRAW(4)	liefert Skalierung der y-Achse (als Fließkommawert)
DRAW(5)	liefert Penflag (–1=PD, 0=PU)

Beispiele:

```
OPENW 0
DRAW "ma 160,128 tt0"        ! starte bei 160,128 mit dem Winkel 0
FOR i%=3 TO 10
  eck(i%,90)                 ! zeichne ein Polygon mit i-Ecken
NEXT i%
'
PROCEDURE eck(n%,r%)         ! n=anzahl der Ecken, r=Kantenlänge
  LOCAL i%
  FOR i%=1 TO n%
    DRAW "fd",r%,"rt",360/n%
  NEXT i%
RETURN
Zeichnet Vielecke.
FOR i=0 TO 320 STEP 8
  SETDRAW 160,100,i
  GRAPHMODE 3
  DRAW "fd 45 rt 90 fd 45 rt 90 fd 45 rt 90 fd 45"
  DRAW "bk 90 rt 90 bk 90 rt 90 bk 90 rt 90 bk 90"
  GRAPHMODE 1
```

```
  DRAW "fd 45 rt 90 fd 45 rt 90 fd 45 rt 90 fd 45"
  DRAW "bk 90 rt 90 bk 90 rt 90 bk 90 rt 90 bk 90"
NEXT i
```

Zeichnet ein kleines und ein großes Rechteck, die sich um ihre eigene Achse drehen.

```
l%=48
' raute 1
DRAW "ma60,100 tt45"
DRAW "fd",l%," rt90 fd",l%," rt90 fd",l%," rt90 fd",l%," rt90"
' rhomboid, schmal
DRAW "mr100,0 tt45"
DRAW "sx0.5 sy0"
DRAW "fd",l%," rt90 fd",l%," rt90 fd",l%," rt90 fd",l%," rt90"
' raute, breit
DRAW "mr100,0 tt45"
DRAW "sx0 sy0.5"
DRAW "fd",l%," rt90 fd",l%," rt90 fd",l%," rt90 fd",l%," rt90"
' raute, breit und hoch
DRAW "mr100,0 tt45"
DRAW "sx3 sy2"
DRAW "fd",l%," rt90 fd",l%," rt90 fd",l%," rt90 fd",l%," rt90"
```

Zeichnet drei Rauten und einen Rhomboiden. Dazu wird als Ausgangsobjekt eine Raute benutzt und die anderen Figuren durch Veränderung der x- bzw. y-Skalierung erreicht.

BOX x1,y1,x2,y2
PBOX x1,y1,x2,y2

x1, y1, x2, y2: iexp

BOX zeichnet ein Rechteck mit den diagonal gegenüberliegenden Eckpunktkoordinaten x1,y1 und x2,y2. Analog dazu zeichnen PBOX ein gefülltes Rechteck.

Beispiel:

```
OPENW 0
BOX 20,20,120,120
x=150
DEFFILL 1,2,4
PBOX 20,20+x,120,120+x
```

Zeichnet eine gefüllte und eine nicht gefüllte Box.

CIRCLE / ELLIPSE

CIRCLE x,y,r
PCIRCLE x,y,r
ELLIPSE x,y,rx,ry
PELLIPSE x,y,rx,ry

x, y, r, rx, ry: iexp

CIRCLE zeichnet einen Kreis um den Mittelpunkt mit den Koordinaten x,y und dem Radius r. Analog dazu zeichnet PCIRCLE einen gefüllten Kreis.

ELLIPSE zeichnet eine Ellipse mit den Mittelpunkt-Koordinaten x,y und dem waagerechten Radius rx sowie dem senkrechten Radius ry. Analog dazu zeichnet PELLIPSE eine gefüllte Ellipse.

Da der Blitter z.Zt. nur Bereiche bis zu 1024 mal 1024 Punkten bearbeiten kann, sind für „rx" und „ry" nur Werte bis zu 503 bzw. 511 erlaubt. Für „r" ist je nach gewählter Auflösung entsprechend ein maximaler Radius vorhanden.

Beispiel:

```
OPENW 0
CIRCLE 160,100,100
ELLIPSE 160,100,50,100
PCIRCLE 160,100,50
PELLIPSE 160,100,100,50
```

Zeichnet einige Kreise und Ellipsen.

POLYLINE n,x(),y() [OFFSET x_off,y_off]
POLYFILL n,x(),y() [OFFSET x_off,y_off]

n, x_off, y_off: iexp
x(), y(): avar-Array

POLYLINE zeichnet einen Linienzug mit n Ecken. Die x,y-Koordinaten der Eckpunkte stehen in den Feldern x() und y(). Der erste Eckpunkt steht in x(0) und y(0), der letzte in x(n–1) und y(n–1). Der erste und der letzte Eckpunkt werden automatisch miteinander verbunden. Zusätzlich kann ein waagerechter (x_off) oder senkrechter (y_off) OFFSET auf diese Koordinaten addiert werden.

POLYFILL füllt das Vieleck mit einem Muster und einer Farbe, die mit DEFFILL festgelegt werden können.

Beispiel:

```
OPENW 0
GRAPHMODE 1
DIM x%(30),y%(30)
DO
  COLOR RAND(64),RAND(64),RAND(64)
  DEFFILL 1,RAND(4),RAND(25)
  FOR i%=0 TO 30
    x%(i%)=RAND(320)
    y%(i%)=RAND(200)
  NEXT i%
  POLYFILL 30,x%(),y%()
LOOP
```

Zeichnet zufällig ausgefüllte Polygone mit jeweils 30 Ecken.

POINT(x,y)

x, y: iexp

Die Farbe des Punktes mit den Koordinaten x,y wird ermittelt und der Variablen zurückgemeldet. Hierbei werden je nach Auflösung, Werte zwischen 0 und 63, zurückgeliefert.

Beispiel:

```
OPENW 0
a=POINT(100,100)
PLOT 100,100
b=,POINT(100,100)
PRINT a,b
```

Schreibt die Farbe an der Fensterkoordinate 100,100 vor und nach dem PLOT-Befehl auf das Fenster.

FILL x,y [,f]

x, y, f: iexp

Dieser Befehl füllt eine beliebig geformte Fläche. Der Füllvorgang beginnt an den Koordinaten x,y. Bei Angabe von „f" wird der Füllvorgang durch Punkte der Farbe f und dem Bildschirmrand begrenzt.

Fehlt „f" so gilt jeder Punkt mit einer anderen Farbe, als der gewählte x/y-Punkt als Begrenzung. Liegt der x/y-Punkt auf einer Begrenzungslinie, wird die Begrenzungslinie gefüllt.

Beispiele:

```
OPENW 0
LINE 0,180,639,180
FOR i=1 TO 19
  BOX i *20,100,i*20+20–i,180
  TEXT i *20–4,195,i
  DEFFILL ,2,i
  FILL i *20+1,101
NEXT i
```

Zeichnet eine Gerade, auf der sich Rechtecke mit verschiedenen Mustern aneinanderreihen und füllt diese ohne Farbbegrenzung, was zur Zerstörung der Muster führt.

```
LINE 0,180,639,180
FOR i=1 TO 19
  BOX i *20,100,i*20+20–i,180
  TEXT i *20–4,195,i
  DEFFILL ,2,i
  FILL i *20+1,101,1
NEXT i
```

Zeichnet darunter ebenfalls eine Gerade mit aufeinanderfolgenden, unterschiedlich gemusterten Rechtecken und füllt mit Farbbegrenzung. Dadurch bleiben die verschiedenen Muster erhalten.

CLS[#n]

n: iexp

Löscht den Bildschirm durch Ausgabe von CHR$(12). Kann auch auf Dateien umgeleitet werden.

Dieser Befehl löscht nur den Textbereich eines Windows. So kann es vorkommen, daß nicht der gesamte Fensterinhalt gelöscht wird. Einer kleiner Bereich des Fensters kann rechts oder unten noch Grafik-Elemente enthalten. Verwenden Sie in diesem Fall besser den Befehl CLEARW.

Beispiel:

```
OPENW 0
PBOX 100,100,500,200
REPEAT
UNTIL MOUSEK
CLS
```

Füllt den Bildschirm teilweise mit einem Rechteck und löscht ihn nach Drücken einer Maustaste.

TEXT x,y,ausdruck

x, y: iexp
ausdruck: sexp oder aexp

Gibt am Punkt mit den Koordinaten x,y den „ausdruck“ als Grafiktext aus. Dieser Punkt bezieht sich auf die linke Ecke der Grundlinie des ersten Zeichens von ausdruck.

Beispiel:

```
OPENW 1
DO
  TEXT MOUSEX,MOUSEY,i%
  INC i%
LOOP
```

Schreibt einen Zählerstand an die Mausposition

GET x1,y1,x2,y2,ausschnitt$
PUT x1,y1,ausschnitt$ [,modus,mask]

x1, y1, x2, y2, modus,mask:iexp
ausschnitt$: svar

GET legt einen Rastport-Ausschnitt (32768 Byte) in einer String-Variablen ab. Die Variablen x1 und y1 legen die Startkoordinate, und x2 bzw. y2 legen die Endkoordinate des Grafikausschnittes fest. Analog zeichnet PUT einen mit GET abgelegten Rastport-Ausschnitt an der Stelle x,y in das aktuelle Fenster. Mit „modus" kann optional bestimmt werden, wie das in „ausschnitt$" abgelegte Bit-Muster mit dem bestehenden Fensterinhalt verknüpft wird.

Im Normalfall kopiert man Bildschriminhalte ohne Veränderung mit einem Wert von 192 (&HC0) für „modus". In der folgenden Tabelle finden Sie die wichtigsten Verknüpfungsregeln:

Modus	Verknüpfungsregel
30	Das Ziel-Bitmuster wird durch das invertierte Quell-Bitmuster ersetzt.
50	Das Ziel-Bitmuster wird invertiert.
60	Exklusiv-Oder-Verknüpfung der Quell- und Ziel-Bitmuster.
80	Nur die in Quell- und Ziel-Bitmuster identischen Bits werden gesetzt.
C0	Das Quell-Bitmuster wird in den Zielbereich kopiert.

Die einzelnen Bits des Parameters „mask" geben die betroffenen Bitplanes an.

VSYNC

Dieser Befehl dient der Synchronisation des Screen-Aufbaus (d.h. das Programm wartet auf senkrechten Strahlrücklauf des Viewport des aktuellen Screen. VSYNC kann z.B. bei der Animation von Bildschirmteilen mit GET und PUT verwendet werden.

Beispiel:

```
OPENW 0
t%=TIMER
FOR i%=1 TO 100
  VSYNC
NEXT i%
PRINT SUB(TIMER,t%)/200
```

Gibt die Zeit für 100 Bildwechsel aus.

DISPLAY ON
DISPLAY OFF

Mit den Befehlen DISPLAY ON und DISPLAY OFF wird der DMA für den Bildaufbau ain- und ausgeschaltet. Nach DISPLAY OFF ist ist das Bild nicht sichtbar bis DISPLAY ON durchgeführt wird. Dadurch ist es möglich, zu zeichnen und das Resulat erst nach dem Zeichnen anzuzeigen. Vor dem Zeichnen muß dazu das DMA-Bit mit DISPLAY OFF gelöscht und nach dem Zeichnen mit DISPLAY ON wieder gesetzt werden.

9 – Ereignis-, Menü-, Screen- und Fensterverwaltung

Ereignisverwaltung

Es gibt spezifische GFA-BASIC-Befehle, mit deren Hilfe man auf einfache Art Ereignisse (Intuition Messages) abfragen kann. Die Überwachung dieser Ereignisse wird durch ein ON MENU xxx GOSUB eingeschaltet, wobei xxx das Ereignis ist, auf das reagiert werden soll.

Die Überwachung findet mit Hilfe des ON MENU-Befehls statt. Bei jedem Aufruf dieses Befehls wird geprüft, ob ein Ereignis eingetreten ist. Ist ein Ereignis vorhanden, auf das reagiert werden soll, so wird zur dafür festgelegten Prozedur verzweigt.

In diesem Abschnitt werden die für GFA-BASIC 3.0 spezifischen Befehle zur Verwaltung von Pull-Down-Menüs eingeführt. Leider herrscht in der Literatur zu diesem Thema eine gewisse Begriffsverwirrung. Daher muß hier zunächst angegeben werden, welche Bedeutungen die damit verbundenen Begriffe in diesem Handbuch haben sollen. Den Begriff Pull-Down-Menü benutzen wir als Oberbegriff. Ein Pull-Down-Menü befindet sich immer in der Titelzeile eines Screens. Wird die rechte Maustaste gedrückt, erscheinen die Überschriften der anwählbaren Menüs. Gelangt der Mauspfeil, bei gedrückter rechter Maustaste, auf eine dieser Überschriften, so klappt darunter ein sogenanntes Menü auf. Jeder Teil dieses Menüs, der einzeln angewählt werden kann, wird als Menüeintrag bezeichnet. Menüeinträge können weitere Menüs erscheinen lassen, wenn man sie mit dem Mauszeiger anwählt. Diese speziellen Menüs werden Untermenüs genannt, ihre Einträge heien entsprechend Untereinträge.

Die Begriffswahl ist nicht allgemein verbindlich, sondern gilt nur für dieses Handbuch; in manchen Büchern werden andere Definitionen verwendet.

Bei der Einrichtung von BASIC-Pull-Down-Menüs werden dessen Einträge in einem String-Array m$() festgelegt. Mit dem Befehl MENU m$() wird dieses Pull-Down-Menü auf den Bildschirm gebracht. Der Befehl ON MENU GOSUB legt eine Prozedur fest, zu der nach der Wahl eines Menüeintrags verzweigt wird.

Während des Programmlaufs wird bei jedem Auftreten eines ON MENU-Befehls geprüft, ob ein Eintrag gewählt wurde.

Mit MENU KILL wird das Pull-Down-Menü abgeschaltet. Der Befehl MENU x,y erlaubt es, Menüeinträge mit Häkchen zu versehen oder in heller Schrift darzustellen, was dazu führt, daß dieser Menüeintrag nicht mehr anwählbar ist.

MENU m$()

m$(): String-Array

Dieser Befehl ist für die Erzeugung eines Pull-Down-Menüs zuständig, wobei sie von den Befehlen und Variablen des vorigen Abschnitts unterstützt werden (ON MENU, MENU()).

MENU m$() stellt das BASIC-Pull-Down-Menü auf dem Bildschirm dar. Im Zeichenkettenfeld m$() stehen die Überschriften und Einträge. Folgendes Format muß bei der Anordnung der Einträge im Feld m$ beachtet werden:

1. Überschrift des Menüs.
2. Liste der Menüeinträge (auch Untereinträge).
3. Ein Leer-String, der das Ende des Menüs markiert.

Hinter dem letzten Menü folgt noch ein weiterer Leer-String, der das Ende des gesamten Pull-Down-Menüs markiert. Ein Menüeintrag, der mit einem Minus-Zeichen beginnt, wird automatisch hell dargestellt und ist nicht anwählbar. Beginnt ein Menüeintrag mit einem Ausrufungszeichen, soll ein Untereintrag erzeugt werden. Folgen nach dem ersten Untereintrag weitere Menüeinträge mit Ausrufungszeichen so werden diese als Untereinträge an den ersten Untereintrag des Menüeintrages angehängt.

MENU(x)

x: aexp zwischen 0 und 11 einschließlich

In den Variablen MENU(0) bis MENU(11) finden sich alle relevanten Informationen für den Umgang mit Ereignissen. Der Aufbau des MENU-Arrays ist ähnlich aufgebaut, wie ein Teil der IntuiMessage-Struktur des Bertriebssystems.

MENU(0)	Menu	Stringarray-Index der BASIC-Menüs
MENU(1)	Class	entspricht IDCMP-Flags
MENU(2)	Code	je nach eingetroffener Message beispielsweise Menü Nummer.
MENU(3)	Qualifier	enthält Nachrichten nach Betätigen von Sondertasten
MENU(4)	IAdress	Adresse von Gadgets oder anderen Objekten
MENU(5)	MouseX	X-Koordinate der Maus relativ zur linken oberen Ecke des aktiven Fensters.
MENU(6)	MouseY	Y-Koordinate der Maus relativ zur linken oberen Ecke des aktiven Fensters.
MENU(7)	Seconds	Seconds und Micros enthalten die
MENU(9)	Micros	Systemzeit zum Zeitpunkt des Ereignisses.
MENU(10)	Windowadress	Die Adresse des Fensters, in dem die Nachricht aufgetreten ist.

Ausgangspunkt der nachfolgenden Erläuterung ist MENU(1). In MENU(1) steht die Kennummer (IDCMP) des aufgetretenen Ereignisses. In Abhängigkeit von MENU(1) enthalten die anderen Elemente des Message-Buffers verschiedene Informationen. Die folgende Tabelle faßt diese Informationen zusammen.

Es ist jeweils der Wert von MENU(1) aufgeführt, dahinter die Bedeutung dieses Wertes und schließlich die Variablen, in denen sich bei diesem MENU(1)-Wert wichtige Informationen befinden.

MENU(1) = 1

Das Sizegadget eines Fensters ist gedrückt worden.
MENU(10): Die Adresse des Fensters.

MENU(1) = 2

Ein Fenster wurde in der Größe verändert.
MENU(10): Die Adresse des Fensters.

MENU(1) = 4

Der Inhalt eines Fenster muß neu gezeichnet werden.
MENU(10): Die Adresse des Fensters.

MENU(1) = 8

Eine Maustaste wurde gedrückt oder losgelassen.
MENU(2): = 104 linke Maustaste gedrückt.
= 232 linke Maustaste losgelassen.
= 105 rechte Maustaste gedrückt.
= 233 rechte Maustaste losgelassen.

MENU(1) = 16

Die Maus wurde bewegt.
MENU(5): X-Koordinate der Maus.
MENU(6): Y-Koordinate der Maus.

MENU(1) = 32

Ein Gadget wurde gedrückt
MENU(4): Adresse des Gadget

MENU(1) = 64

Ein Gadget wurde losgelassen
MENU(4): Adresse des Gadget

MENU(1) = 128

In einem Fenster wurde ein Requester gezeichnet.
MENU(10): Die Adresse des Fensters.

MENU(1) = 256

Die Menütaste der Maus wurde zur Menüauswahl gedrückt

MENU(2): angewähltes Menü (Bitfeld).
Die Entschlüsselung des Bitfeldes geschieht folgendermaßen:
Menünummer = MENU(2) AND 31
Menüeintrag = SHR((MENU(2),5) AND 63
Untereintrag = SHR((MENU(2),11) AND 31

Bei BASIC-Menüs enthält MENU(0) den Index des angewählten Menüpunkts innerhalb des Stringarrays.

MENU(1) = 512

Das Schließfeld eines Fenster wurde gedrückt.

MENU(10): Adresse des Fensters.

MENU(1) = 1024

Eine Taste wurde gedrückt oder losgelassen.

MENU(2): Scan-Code der Taste. (siehe Scan-Codes im Anhang)

MENU(1) = 2048

Das Fenster ist nicht bereit, einen Requester anzuzeigen.

MENU(10): Adresse des Fensters.

MENU(1) = 4096

Wenn der letzte Requester des Fensters geschlossen wird.

MENU(10): Adresse des Fensters.

MENU(1) = 8192

Das Fenster ist nicht bereit, eine Menüleiste anzuzeigen.

MENU(10): Adresse des Fensters.

MENU(1) = 16384

Die Systempreferences wurden verändert.

MENU(1) = 32768

Eine Diskette wurde in ein Laufwerk eingelegt.

MENU(1) = 65536

Eine Diskette wurde aus einem Laufwerk genommen.

MENU(1) = 131072

Auf der Workbench wurde irgendeine Veränderung vorgenommen.

MENU(1) = 262144

Das Fenster wurde mit der Maus aktiviert.

MENU(10): Adresse des Fensters.

524288

MENU(1) = 262144

Das Fenster wurde deaktiviert.

MENU(10): Adresse des Fensters.

MENU(1) = 1048576

Die Maus wurde bewegt.

MENU(5): Delta X-Koordinate
MENU(6): Delta Y-Koordinate

MENU(1) = 2097152

Eine Taste wurde gedrückt.

MENU(2): ASCII-Code der Taste je nach eingestellter Keymap.

MENU(1) = 4194304

Timer Nachricht (jede 1/10 Sekunde)

Die Werte für MENU(7) bis MENU(10) werden unabhängig von der eingetretenen Nachricht immer gesetzt.

ON MENU [t]
SLEEP

t: iexp

Der Befehl ON MENU überwacht das Auftreten von Ereignissen. Vor diesem Befehl sollte festgelegt werden, in welche Prozedur das Programm bei welchem Ereignis verzweigen sollte. Die Befehle, mit denen dies festgelegt wird, werden im Rest dieses Kapitels beschrieben. Für ein ständiges Überwachen eines Ereignisses ist es notwendig, diesen Befehl wiederholt zu durchlaufen. Daher findet sich der ON MENU-Befehl normalerweise in einer Schleife.

Der Parameter t enthält die Zeit (in Intui-ticks=Zehntel Sekunden), nach der der ON MENU-Befehl beendet sein soll.

SLEEP ist gleichbedeutend mit ON MENU 1 und wartet auf einen Intui-Tick oder ein anderes Ereignis (Menü-Auswahl o.ä.). Intui-ticks (Zeitmeldungen des Intuition) erfolgen nur bei aktiviertem Fenster.

ON MENU GOSUB proc
ON MENU KEY GOSUB proc
ON MENU BUTTON GOSUB proc
ON MESSAGE GOSUB proc

proc: Name einer Prozedur

Wenn eine Nachricht eintrifft, so verzweigt das Programm in die Prozedur mit dem Namen „proc". Die Abfragen erfolgen bei ON MENU bzw. SLEEP. Der Aufbau der Message-Struktur wird im Abschnitt über MENU(x) besprochen. Die Nachricht, die dabei auftreten kann, ist abhängig von den beim OPENW gesetzten IDCMP-Flag des Fensters. Wird also bei einem OPENW-Befehl mit Parameter, in den IDCMP-Flags beispielsweise MENUPICK gesetzt, so wird nach Auftreten einer Menünachricht in die Prozedur „proc" verzweigt.

Mit ON MENU GOSUB proc wird eine Prozedur festgelegt, zu der nach Anwahl eines Menüeintrags verzweigt wird. Innerhalb der Prozedur kann mit Hife der Variable MENU(0) herausgefunden werden, welcher Menüeintrag gewählt wurde.

Mit ON MENU KEY GOSUB proc wird eine Prozedur zur Behandlung von Tastatureingaben festgelegt, Tastencode in MENU(2). Mit ON MENU BUTTON GOSUB proc wird entsprechend die Prozedur zur Behandlung von Mausknopf-Betätigungen festgelegt.. In MENU(2) steht eine Kennung für die gedrückte oder losgelassene Maustatste.

Beispiel:

```
DIM a$(100)                    !Menu einlesen
DO
  READ a$(i%)
  EXIT IF a$(i%)="*"
  INC i%
LOOP
a$(i%)=""
DATA main,  about,  info,  test,
DATA file,  load,  save,----------,  quatsch,
DATA *
OPENW 2,0,100,640,120,&H408,1        !RawKey und Buttons
MENU a$()
```

```
MENU 3," TEST"                          !Text für Eintrag 3 ändern
MENU KEY 6,65                           !Amiga-A für Load
ON MENU GOSUB men                          !Routinen festlegen
ON MESSAGE GOSUB msg
ON MENU BUTTON GOSUB but
ON MENU KEY GOSUB key
DO
  SLEEP                                 !Warten
LOOP
PROCEDURE men                              !bei Menuauswahl
  PRINT MENU(0)'a$(MENU(0))
RETURN
PROCEDURE msg                              !irgendwelche Messages
  PRINT "msg:"'
  FOR i%=0 TO 9
    PRINT HEX$(MENU(i%),8)'
  NEXT i%
  PRINT
RETURN
PROCEDURE but                              !MouseButtons
  PRINT "but:"'
  SELECT MENU(2)
  CASE &H68
    PRINT "linke Maustaste gedrückt"
  CASE &HE8
    PRINT "linke Maustaste losgelassen"
  CASE &H69       !kommt nicht wegen Menu
    PRINT "rechte Maustaste gedrückt"
  CASE &HE9       !kommt nicht wegen Menu
    PRINT "rechte Maustaste losgelassen"
  ENDSELECT
RETURN
PROCEDURE key                              !RawKeys (siehe Anhang)
  PRINT "key:"'HEX$(BCLR(MENU(2),7),2)'
  IF BTST(MENU(2),7)
    PRINT "losgelassen"
  ELSE
    PRINT "gedrückt"
  ENDIF
RETURN
```

Es wird auf Ereignisse gewartet und von den eintretenden Ereignissen jeweils nur die Mausereignisse abgefragt. Das Programm kann durch Drücken der rechten Maustaste beendet werden.

MENU KILL
MENU x,y
MENU x,str$
MENU KEY x,y

x, y: aexp
str$: sexp

MENU KILL schaltet das Pull-Down-Menü ab. Außerdem schaltet MENU KILL die ON MENU GOSUB-Einstellungen ab. Mit dem Befehl MENU x,y kann der x-te Eintrag eines BASIC-Menüs umge-stellt werden. Die Numerierung der Einträge entspricht den Indizes des String-Arrays, in dem die Pull-Down-Menüeintrage stehen. Die Zählung beginnt also bei Null; Überschriften und Leer-Strings werden mitgezählt.

Der zweite Parameter y gibt an, was mit dem x-ten Menüeintrag getan werden soll. Dabei gilt folgende Zuordnung:

y	Effekt
64	Menüeintrag wird bei Anwahl invers angezeigt.
128	Ein Kasten wird um das Menü gezeichnet.
192	Das Menü soll nicht veränderbar sein.
256	Häkchen davor setzen.

Mit „str$“ kann ein neuer Text für den Menü-Eintrag zugewiesen werden. Die neue Zeichenkette darf allerdings nicht länger sein, als der bisherige Text; alle weiteren Zeichen werden nämlich ignoriert.

Der Befehl MENU KEY ordnet einem Menü-Eintrag ein ASCII-Zeichen zu (Shortcut). Wird dieses zusammen mit der rechten Amiga-Taste gedrückt, entspricht dies der Anwahl des zugeordneten Menü-Eintrags. „x“ ist der Index auf den Eintrag und „y“ enthält den ASCII-Wert. Ist „y“ Null, wird die Zuweisung gelöscht. Ein Shortcut wird im Menü durch das Amigatasten-Symbol und den ASCII-Wert dargestellt.

Beispielprogramm für den Abschnitt Pull-Down-Menü:

```
DIM eintrag$(20)
i%=-1
REPEAT
  INC i%
  READ eintrag$(i%)
UNTIL eintrag$(i%)="Schlußmarke"
eintrag$(i%)=""
eintrag$(i%+1)=""
MENU eintrag$()
ON MENU GOSUB auswertung
OPENW 0
DATA Desk, Test ,
DATA File, Load , Save ,———, Quit ,
DATA Titel, Eintrag 1 , Eintrag 2 ,
DATA Schlußmarke
REPEAT
  ON MENU
UNTIL MOUSEK=2
PROCEDURE auswertung
  ' MENU(0) enthält den Array-Index des angewählten Eintrags
  m%=MENU(0)
  PRINT eintrag$(m%)
  ALERT 0,"Häckchen davor ?",1," Ja | Nein ",a%
  IF a%=1
    MENU m%,257   ! Abhaken
  ELSE
    MENU m%,64    ! Invertieren
  ENDIF
  ALERT 0,"Helle Schrift ?|(nicht wählbar)",1," Ja | Nein ",a%
  IF a%=1
    MENU m%,16
  ELSE
    MENU m%,64
  ENDIF
RETURN
```

Erzeugt und verwaltet ein Pull-Down-Menü. Beim Anklicken eines Menüeintrags wird dessen Text auf den Monitor geschrieben und gefragt, wie der gewählte Menüeintrag dargestellt werden soll.

Screen-Befehle

Durch die sehr komfortablen Screen-Befehle von GFA-BASIC können alle Grafikauflösungen des Amiga eingestellt werden. GFA-BASIC beinhaltet die Befehle OPENS, CLOSES, MOVES, FRONTS, BACKS und TITELS sowie die Funktion SCREEN — mit allen Möglichkeiten, die Screens bieten.

Screens dienen als Grundlage für alle grafischen Ausgaben. Es sind virtuelle Bildschirme, die unabhängig voneinander verschiedene Auflösungen haben können.

Beim Öffnen von Screens werden in GFA-BASIC Indizes von 1 bis 31 vergeben, was das Screenhandling sehr vereinfacht. Für den Workbench-Screen gilt der Index 0. Wird kein Screen geöffnet, so finden alle Ausgaben auf dem Workbench-Screen statt. Auf den Workbench-Screen sind nur die Befehle FRONTS und BACKS anwendbar!

Die Anzahl der Screens, die geöffnet werden können, ist demnach nur durch die Indizes und den Grafikspeicher begrenzt. Der Speicherbedarf pro Screen ist abhängig von der eingestellten Größe und Bitplanetiefe. Ein Screen mit einer Bitplane benötigt weniger Speicherplatz als ein Screen mit fünf Bitplanes. Allerdings muß hierbei bedacht werden, daß auf einem Screen mit weniger Bitplanes die Anzahl der darstellbaren Farben abnimmt.

Die maximale Größe eines Screen liegt bei 1024 Pixel in der Breite und Höhe. Auf dem Amiga Monitor kann nur der Bereich bis 640 Pixel in der Breite und 512 Pixel in der Höhe, ab der linken oberen Ecke des Screens, dargestellt werden.

Für Screens können Sie folgende Auslösungen eingestellen:

HIRES 32768 (&H8000)

Im HIRES-Modus sind maximal vier Bitplanes erlaubt. Der sichtbare Bereich ist 640 Pixel in der Breite und 256 in der Höhe. Die Anzahl der Farben die dargestellt werden können, richtet sich nach der im Befehl angebenen Anzahl der Bitplanes. Hierbei gilt: 2 ^ Anzahl der Bitplanes.

Für den HIRES-Modus sind dies maximal 16 verschiedene Farben.

LACE 4 (&H4)

Im LACE-Modus wird ein spezielles Verfahren eingeschaltet, das die Zeilenanzahl verdoppelt. Jedes Bild wird aus zwei Halbbildern zusammengesetzt. Da der Monitor mit einer Bildfrequenz von 50 Hz arbeitet, und ein kompletter Bildaufbau in diesem Modus 1/30 Sekunde dauert, flackert das Bild etwas.Die sichtbare Breite und Höhe beträgt hier 320 * 512 Pixel. Bei 5 Bitplanes, die in diesem Modus erlaubt sind, können 32 verschiedene Farben angezeigt werden.

Kombiniert man den Modus LACE mit HIRES (32772 (&H8004)) wird zusätzlich noch die horizontale Auflösung verdoppelt. Da Hires nur vier Bitplanes zuläßt sind auch in der Kombination mit Lace nur vier Bitplanes zulässig.

EXTRA_HALFBRITE 128 (&H80)

Mit maximal sichtbaren 320 * 256 Pixel Auflösung in diesem Modus können sechs Bitplanes und somit 64 verschiedene Farben angezeigt werden.

HAM 2048 (&H800)

In diesem speziellen Auflösungsmodus ist es möglich, 4096 Farben gleichzeitig anzuzeigen. Dies geschieht über einen Trick, denn nach der Rechnung ergibt 2^6 Bitplanes = 64 Farben. Hold and modify (HAM) bedeutet, nimm ein Pixel und modifiziere es mit der Pixelfarbe des voranstehenden Pixels.

Der Auflösungsmodus hat nur dann einen Sinn, wenn mindestens fünf Bitplanes im OPENS-Befehl angegeben werden. Zeichnet man in diesem Modus mit den Farben der ersten 16 Farbregister (0–15), tritt keine sichtbare Veränderung gegenüber den anderen Auflösungsmodi ein.

Erst wenn Farben der Register 16–63 als Zeichenfarbe gesetzt werden, treten die Eingenschaften dieses Auflösungsmodus in Erscheinung.

Entspricht die Kombination der Bits von Bitplane fünf und sechs gleich %00, so werden die Farben nicht verändert. Dies ist der Fall, wenn mit Farben der Register 0 – 15 gezeichnet wird.

Entspricht die Bitkombination %01, %10 oder %11, wird die Farbe des unmittelbar links davon stehenden Pixels kopiert und verändert. Hierbei wird die Bitkombination von Bitplane 1–4 anstelle der jeweiligen vier Farbbits eingesetzt, ohne die Farben des entsprechenden Farbregisters zu ändern.

Dabei gilt für:

%01 = Ersetzen der Blaubits.
%10 = Ersetzen der Rotbits.
%11 = Ersetzen der Grünbits.

In Kombination mit den Auflösungsmodi können noch zusätzlich folgende Einstellungen gemacht werden:

GENLOCK_VIDEO 2 (&H2)
Über ein Genlock-Video-Interface kann die Screen-Hintergrundfarbe durch Videobilder ausgetauscht werden.

SPRITES 16384 (&H4000)
Sollen Hardware-Sprites erzeugt werden können, muß dieses Flag gesetzt sein.

OPENS nr [,x, y, w, h, bitp, mod]

nr, x, y, w, h, bitp: aexp
mod: iexp

Der Befehl OPENS öffnet einen Screen.

In „nr" können Indexnummern von 1 bis 15 angegeben werden. Die Indexnummer 0 steht für den Workbench-Screen. Folgen dem Index keine weiteren Parameter so wird ein Standard-Screen geöffnet. Die Screen-Größe, Bitplanetiefe und Auflösungsmodi werden dann auf Defaulwerte gesetzt.

Führt man den Befehl OPENS mehrmals mit dem gleichen Screen-Index aus wird der Screen zum aktuellen Ausgabe-Screen gemacht. Weitere Parameter werden dann ignoriert.

x und y enthalten die Startkoordinate des Screens. Die Angabe von x (Abstand zur linken Bildschirmseite) wird beim Öffnen des Screens nicht berücksichtigt, ist aber für spätere Betriebsystem-Versionen schon vorgesehen.

Die Werte für die Weite w und Höhe h sind relativ zur linken oberen Ecke des Screens. Werden für x, y, w oder h negative Werte angegeben, erscheint eine Fehlermeldung.

Vorangegangen finden sie die Beschreibung der Auflösungsmodi, die im Ausdruck „mod" angegeben werden können. „mod" kann folgende Werte annehmen:

Wert	Name	sichtbarer Bereich	Bitplanes
32768	HIRES	640 * 256[1]	4
32772	HIRES LACE	640 * 512[1]	4
2048	HAM	320 * 256[1]	6
128	EXTRA_HALFBRITE	320 * 256[1]	6
4	LACE	320 * 512[1]	5
0	STANDARD	320 * 256[1]	5

[1] Bei NTSC-Auflösung (USA) 200 bzw. 400.

In „bitp“ steht die Anzahl der Bitplanes, die für den Screen gesetzt werden sollen. Die Anzahl der Bitplanes in der Tabelle bedeutet jeweils die maximale Anzahl, die man setzen kann.

Beispiel:

```
OPENS 1,0,0,320,256,6,&H800          ' öffnet Screen im HAM–Modus
OPENW 0,0,0,320,256,0,0
GRAPHMODE 0
FOR r%=0 TO 15
  FOR g%=0 TO 15
    FOR b%=0 TO 15
      x%=r%    *20
      y%=g%+(b%    *16)
      COLOR r%+&H30
      LINE x%,y%,x%+19,y%
      INC x%
      COLOR g%+&H20
      LINE x%,y%,x%+19,y%
      INC x%
      COLOR b%+&H10
      LINE x%,y%,x%+19,y%
    NEXT b%
  NEXT g%
NEXT r%
```

Es wird ein Screen im HAM-Modus geöffnet und eine HAM-Farbpalette angezeigt.

CLOSES nr

nr: aexp

Der Befehl CLOSES schließt einen geöffneten Screen und gibt den Speicherbereich des Screens wieder frei. „nr“ ist der Index des Screens. Existiert der Screen mit dem Index „nr“ nicht, erhält man eine Fehlermeldung. CLOSES schließt alle noch geöffneten Fenser auf dem Screen, bevor der Screen geschlossen wird.

Beispiel:

```
OPENS 1
I%=1
DO
  MOVES 1,0,I%
  INC I%
  EXIT IF I%=200
LOOP
```

Es wird ein Screen geöffnet, verschoben und wieder geschlossen.

FRONTS nr
BACKS nr

nr: aexp

Jeder geöffnete Screen enthält an seinem rechten oberen Rand zwei Gadgets. Über diese Gadgets ist es möglich, den Screen in seiner Lage zu verändern. Betätigt man eines der beiden Gadgets mit der Maus, so wird der Screen im Vordergrund angezeigt oder in den Hintergrund verlagert. Das Gleiche kann auch durch die Befehle FRONTS und BACKS ausgeführt werden. FRONTS zeigt den Screen mit dem Index „nr" im Vordergrund an. BACKS schiebt den Screen in den Hintergrund.

Beispiel:

```
FOR i%=1 TO 5
  OPENS i%,0,i%*30,320,200,1,128
NEXT i%
DO
  FRONTS RAND(5)
  BACKS RAND(5)
LOOP
```

An verschiedenen Positionen werden fünf Screens geöffnet und in einer Endlosschleife zufällig im Vordergrund angezeigt oder in den Hintergrund gestellt.

MOVES nr,x,y

nr: aexp

In dem Befehl OPENS wird in x und y die Position angegeben, ab welcher der Screen auf dem Bildschirm gezeichnet werden soll. Mit dem Befehl MOVES können Sie die Position nachträglich noch ändern. Im Gegensatz zur Betriebssystemfunktion MoveScreen() wird in x und y die absolute Koordinate verlangt. Die Koordinate bezieht sich auf die linke obere Ecke des Bildschirmes. Achten sie bitte darauf, daß die Werte in x und y nicht negativ werden, da sonst eine Fehlermeldung ausgegeben wird.

Ist während der Ausführung des Befehls MOVES ein Window auf dem Screen dargestellt, kann der Mauszeiger nicht bewegt werden. Dies verhindert unnötige Fehler des Betriebssystems, wenn ein Window in einem bewegten Screen verschoben wird.

Beispiel:

```
OPENS 1
OPENS 2
OPENW 0
FRONTS 1
FOR x& = 1 TO 2
  FOR i&=0 TO 100
    MOVES x&,0,i&
  NEXT i&
NEXT x&
```

Es werden zwei Screens geöffnet und verschoben.

TITLES nr,str

nr: aexp
str: sexp

Jeder Screen kann in der obersten Zeile einen Titel enthalten. Wird ein Screen nur mit dem Parameter Index geöffnet, so wird in die Titelzeile der voreingestellte Titel für Screens übernommen.

Mit dem Befehl TITLES kann der Titel nach Öffnen des Screens geändert werden. Die in „str" angegebene Zeichenkette wird in die Titelzeile des Screens mit dem Index „nr" eingetragen. Enthält „str" einen Leerstring so wird der vorhandene Screen-Titel gelöscht.

Beispiel:

```
OPENS 1
a$="NEUER TITEL"
TITLES 1,a$
```

Der Screen erhält nach dem Öffnen einen neuen Titel.

SETSPEN detail_pen,block_pen

detail_pen, block_pen: iexp
title$: sexp

Die beiden Befehle können zur Voreinstellung von Screens benutzt werden. Sie ersetzen die entsprechenden Werte aus den Voreinstellungen. In „detail_pen“ wird die Farbnummer der Farbe eingesetzt, in der der Screentiteltext erscheinen soll. An „block_pen“ wird die Farbnummer des Screen-Titelfeldes zugewiesen.

SCREEN(nr)
SCREEN(adr)

nr, adr: iexp

Die Funktion SCREEN liefert bei Angabe des Screen-Indexes „nr" die Adresse der Screen-Struktur und umgekehrt, bei Angabe der Adresse „adr" den Index des Screens zurück.

Beispiel:

```
OPENS 1
adr%=SCREEN(1)
index%=SCREEN(adr%)
PRINT adr%,index%
```

Die erste SCREEN-Funktion ermittelt die Adresse, über die in der zweiten Anweisung der Index des Screens erfragt wird.

Fenster sind Bildschirmbereiche die je nach Art vergrößert, verkleinert und verschoben werden können. GFA-BASIC beinhaltet die Befehle OPENW, CLOSEW, CLEARW, TITLEW, FULLW, MOVEW, SIZEW, LIMITW, FRONTW, BACKW sowie die Funktion WINDOW(), die eine sehr komfortable Fensterverwaltung ermöglichen.

In GFA-BASIC können Grafik- und Textausgaben generell nur auf dem aktuellen Rastport oder Fenster erfolgen. Stößt der Interpreter auf einen Befehl, der eine Bildschirmausgabe erzeugt, und es wurde zuvor kein Fenster geöffnet, wird automatisch das Fenster mit dem Index 0 auf dem aktuellen Screen geöffnet.

Für Fenster werden Indizes von 0 bis 15 vergeben.

OPENW nr
OPENW nr [,x, y, w, h, IDCMP, flags]
OPENW nr [,x, y, w, h, IDCMP, flags,scrnr]
OPENW nr [,x, y, w, h, IDCMP, flags,scrnr,bitmap]
CLOSEW nr

nr, x, y, w, h, scrnr, IDCMP, bitmap: iexp

OPENW öffnet das Fenster mit dem Index „nr". Werden außer dem Fensterindex keine weiteren Parameter angegeben so wird ein Standard-Fenster geöffnet. Fenster 0 belegt dann den gesamten Screen mit Ausnahme der Titelzeile, Fenster 1 bis 15 haben jeweils ein Viertel der Screengröße. #1 liegt links oben, #2 rechts oben, #3 links unten, #4 rechts unten und #5 bis #15 liegen zentriert.

„x" und „y" enthalten die Ursprungskoordinate relativ zur linken oberen Ecke des Screens. In „w" und „h" wird die Weite und Höhe des Fenster angegeben. Negative Werte oder Werte, die außerhalb des Screen-Bereiches liegen, erzeugen eine Fehlermeldung.

IDCMP heißt „Intuition Direct Communication Message Port". Jedes Fenster hat einen eigenen Message Port, über den Ereignisse empfangen werden, sobald das Fenster aktiv ist. Durch IDCMP wird spezifiziert, welche Nachricht in den Message Port gelangen soll. Setzt man für IDCMP den Wert von MOUSEBUTTONS, enthält der Message Port immer, wenn eine Maustaste gedrückt wurde, den Wert von MOUSEBUTTONS. Es ist dann sehr einfach, die Nachrichten zu selektieren und in seinem Programm entsprechend zu reagieren.

Man kann folgende Ereignisse empfangen:

1 (&H1) SIZEVERIFY

Das Sizegadget wurde betätigt. Das ist besonders wichtig, wenn Ausgaben in einem Fenster stattfinden, die unbedingt vor einer Größenänderung des Fensters beendet sein müssen.

2 (&H2) NEWSIZE

Wurde das Fenster in der Größe verändert, erhält man die Nachricht NEWSIZE.

4 (&H4) REFRESHWINDOW

Der Fensterinhalt soll erneuert werden. Refreshwindow gilt nur, wenn für den Refresh-Typ des Fensters SMART_REFRESH oder SIMPLE_REFRESH gesetzt wird.

8 (&H8) MOUSEBUTTONS

Eine Maustaste wurde betätigt. Dies gilt nur bedingt für die rechte Maustaste, da diese die Menüleiste einschaltet. Ist das RMBTRAP-Flag in den Fensterflags gesetzt, kann die Menüleiste nicht mehr eingeschaltet werden und man erhält ebenfalls für das Betätigen der rechte Maustaste eine MOUSEBUTTONS-Nachricht.

16 (&H10) MOUSEMOVE

Ist das REPORTMOUSE-Flag in den Fensterflags gesetzt, wird diese Nachricht gesendet, sobald die Maus bewegt wird.

32 (&H20) GADGETDOWN

Bei Selektierung eines Gadgets mit der Maus. Dies gilt nicht für die Standard-Fenster-Gadgets.

64 (&H40) GADGETUP

Ein Gadget wurde losgelassen. Dies gilt ebenfalls nicht für die Standard-Fenster-Gadgets.

128 (&H80) REQSET

Der erste Requester im Fenster wurde geöffnet.

4096 (&H1000) REQCLEAR

Der letzte Requester im Fenster wurde gelöscht.

256 (&H100) MENUPICK

Die rechte Maustaste wurde zur Menüauswahl losgelassen.

512 (&H200) CLOSEWINDOW

Klick auf das Schließfeld des Fensters.

1024 (&H400) RAWKEY

Eine Taste wurde gedrückt. Es werden die normalen Tastencodes übermittelt.

2097152 (&H200000) VANILLAKEY

Eine Taste wurde gedrückt. Es werden entsprechend der eingestellten Keymap ASCII-Codes übermittelt.

262144 (&H40000) ACTIVEWINDOW

Bei Aktivierung des Fensters.

524288 (&H80000) INACTIVEWINDO

Bei Inaktivierung des Fensters

1048576 (&H100000) DELTAMOVE

Deltamove steht in Verbindung mit dem IDCMP Mousemove. Wird die Maus bewegt, so werden die Mauskoordinaten relativ zur Startposition der Maus übermittelt.

4194304 (&H400000) INTUITICKS

Es wird jede 1/10 Sekunde eine Nachricht gesendet.

16384 (&H4000) NEWPREFS

Die System-Preferences wurden geändert.

32768 (&H8000) DISKINSERTED

Bei Einlegen einer neuen Diskette in das Laufwerk.

65536 (&H10000) DISKREMOVED

Die Diskette wurde aus dem Laufwerk entfernt.

Die letzten drei Nachrichten erhält man nur, wenn das Fenster nicht aktiv ist. Die Angaben, die in „flags" gemacht werden, bestimmen das Aussehen des Fensters. Für „flags" können folgende Werte angegeben werden:

1 (&H1)WINDOWSIZING

Systemgadget zum Vergrößern oder Verkleinern von Fenstern.

16 (&H10) SIZEBRIGHT

Das Gadget wird an der rechten Seite angebracht.

32 (&H20) SIZEBOTTOM

Das Gadget wird am unteren Rand angebracht. (Volle 80 Zeichen)

2 (&H2) WINDOWDRAG

Das Fenster kann verschoben werden.

4 (&H4) WINDOWDEPTH

Das Fenster kann in den Hinter- oder Vordergrund geklickt werden.

8 (&H8) WINDOWCLOSE

Systemgadget zum Schließen des Fensters.

0 (&H0) SMARTREFRESH

Der sichbare Fensterinhalt wird in einen reservierten Speicherbereich kopiert. Wird das Fenster von anderen Fenstern überlappt, so kann der Fensterinhalt wieder hergestellt werden. Nicht aber bei einer Größenveränderung des Fensters.

64 (&H40)SIMPLEREFRESH

Muß der Fensterinhalt erneuert werden, so wird die Nachricht Refreshwindow an den Message Port übermittelt.

128 (&H80) SUPERBITMAP

Es wird eine eigene Bitmap verwendet, in der der ganze Fensterinhalt gespeichert ist. Dadurch ist gewährleistet, daß bei Vergrößerung des Fensters der Inhalt nicht erneuert werden muß. Die Initialisierung der Bitmap muß vom Programmierer selbst übernommen werden.

131072 (&H20000) NOCAREREFRESH

Die Nachricht zum Erneuern des Fensters wird nicht gesendet.

256 (&H100) BACKDROP

Das Fenster kann nicht vor andere gelegt werden.

512 (&H200) REPORTMOUSE

Die Mausposition soll übermittelt werden.

1024 (&H400) GIMMEZEROZERO

Die Randkomponenten und der Fensterinnenteil werden getrennt gespeichert, so daß nicht über den Fensterrand gezeichnet werden kann. Die Zeichenkoordinaten sind relativ zur linken oberen Ecke des Fensterinnenbereiches.

2048 (&H800) BORDERLESS

Es werden keine Randkomponenten gezeichnet.

4096 (&H1000) ACTIVATE

Das Fenster wird beim Öffnen aktiviert.

65536 (&H10000)RMBTRAP

Für die rechte Maustaste soll die Nachricht MOUSEBUTTONS übermittelt werden. „scrnr" enthält den Screen-Index des Screens, auf dem das Fenster ausgegeben werden soll. Wird OPENW ohne diesen Paramenter ausgeführt, so wird das Fenster auf dem aktuellen Screen geöffnet. Sie sollten darauf achten, in „scrnr" den Index eines bereits geöffneten Screens anzugeben, da sonst während des Programmlaufes eine Fehlermeldung ausgegeben wird.

Soll ein Fenster erzeugt, werden dessen Inhalt in einer eigenen Bitmap zwischengespeichert wird, muß „bitmap" die Adresse der Bitmap enthalten. Ist in „flags" das Fenster-Flag SUPERBITMAP gesetzt wird davon ausgegangen, daß eine eigene Bitmap zum Erneuern des Fensterinhaltes angelegt wurde. Die Bitmap muß mindestens so groß sein wie die maximale Größe des Fensters.

Der Befehl CLOSEW schließt das Fenster mit dem Index nr. Ist das Fenster mit dem Index nr bereits geschlossen wird eine Fehlermeldung ausgegeben.

Beispiel:

```
OPENS 1
OPENS 2
OPENW 0
OPENW 1,0,0,100,200,0,15
OPENW 2,0,0,100,200,0,15,1
```

Es werden zwei Screens und drei Fenster geöffnet. Die ersten beiden Fenster stehen dann auf dem zuletzt geöffneten Screen und das zuletzt geöffnete Fenster wird auf dem zuerst geöffneten Screen angezeigt.

CLEARW nr
TITLEW nr,a$[,b$]
FRONTW nr
BACKW nr
FULLW nr

nr: iexp
a$,b$: sexp

Der Befehl CLEARW löscht den Inhalt des Fenster mit dem Index „nr". TITELW schreibt in die oberste Zeile des Fensters „nr" den Text a$. Ist b$ angegeben, wird dieser, wenn das Fenster aktiv ist, in der Titelzeile des Screens angezeigt. FRONTW holt das Fenster mit dem Index „nr" in den Vordergrund. BACKW schiebt das Fenster in den Hintergrund. FULLW bringt das Fenster „nr" auf Screengröße.

Beispiel:

```
DEFFILL 1,2,4
PBOX 0,0,639,399
OPENW 1
PAUSE 50
FULLW 1
PRINT "Fenster 1"
OPENW 4,0,0,100,100,0,&HF
PAUSE 50
CLEARW 1
OPENW 3
PAUSE 50
FRONTW 1
PAUSE 50
CLOSEW 1
TITLEW 4,"Fenster 4"
PAUSE 100
CLOSEW 3
CLOSEW 4
CLOSEW 0
```

Läßt einige Fenster erscheinen, verändert sie und schließt sie wieder.

MOVEW nr,x,y
SIZEW nr,w,h
LIMITW nr,min_x,min_y,max_x,max_y

nr, x, y, w, h, min_x, min_y, max_x, max_y: aexp

MOVEW verschiebt das Fenster mit dem Index „nr" auf die absolute x- und y-Position relativ zur linken oberen Screen-Ecke. Mit SIZEW kann das Fenster „nr" vergrößert oder verkleinert werden. In w und h wird die neue Weite und Höhe eingegeben. LIMITW setzt die Minimal- und Maximalgröße eines Fensters. In „min_x" und „min_y" muß die Minimalgröße und in „max_x", max_y die Maximalgröße des Fenster eingegeben werden. Dies gilt für die Veränderung der Fenstergröße mit der Maus.

Beispiel:

```
OPENS 1
OPENS 2
OPENW 0,0,0,160,100,0,15,1
OPENW 1,0,0,160,100,0,15,2
LIMITW 0,100,100,200,200
FULLW 0
FULLW 1
```

Es wird jeweils ein Fenster in einem Screen ausgegeben und mit dem Befehl FULLW auf die maximale Größe gebracht.

SETWPEN detail_pen,block_pen

detail_pen, block_pen: iexp
title$: sexp

SETWPEN setzt wie der entsprechende Screen-Befehl die Voreinstellungen neu. In „detail_pen" wird die Farbnummer der Farbe eingesetzt, in der der Fenstertiteltext erscheinen soll. An „block_pen" wird die Farbnummer des Fenstertitelfeldes zugewiesen.

WINDOW(nr)
WINDOW(adr)

nr,adr: iexp

Die Funktion WINDOW liefert bei Angabe des Fensterindexes „nr“ die Adresse der Window-Struktur und umgekehrt bei Angabe der Adresse „adr“ den Index des Fensters zurück.

Beispiel:

```
OPENW 1
adr%=WINDOW(1)
index%=WINDOW(adr%)
PRINT adr%,index%
```

Die erste WINDOW Funktion ermittelt die Adresse, über die in der zweiten Anweisung der Index des Fensters erfragt wird.

ALERT sym,text$,default,button$,wahl

sym, default: iexp
text$, button$: sexp
wahl: avar

Der Befehl ALERT läßt einen Alert-Requester auf dem Bildschirm erscheinen. Der Ausdruck „sym" ist für spätere GFA-BASIC-Versionen vorgesehen und wird vom Interpreter nicht berücksichtigt.

Im String-Ausdruck text$ steht der Text, der im Alert-Requester erscheinen soll. Es sind maximal fünf Zeilen mit jeweils höchstens 40 Zeichen erlaubt. Die Zeilen werden durch ein | voneinander getrennt. Zeilen, die länger als 40 Zeichen sind, werden abgeschnitten.

Der Ausdruck „default" gibt an, der wievielte Auswahlknopf des Alert-Requesters das sogenannte Default-Gadget ist. Dieser Auswahlknopf kann nicht nur mit der Maus, sondern auch durch Drücken der Return- oder Enter-Taste gewählt werden. Existiert kein Gadget mit der Nummer default, so kann der Alert-Requester nicht mit Return oder Enter verlassen werden, sondern nur mit der Maus.

Der String-Ausdruck gadget$ enthält den Text der Gadgets. Die maximale Textlänge pro Gadget ist 40 Zeichen. Die Texte der einzelnen Gadgets werden durch | getrennt. Dabei können maximal drei Gadgets dargestellt werden. In der Variable „wahl" wird die Nummer des gewählten Gadgets zurückgemeldet.

Beispiel:

```
ALERT 0,"Wählen Sie|ein Gadget !",1,"Links|Rechts",a%
ALERT 0,"Sie haben Gadget "+STR$(a%)+" gewählt.",0," Ok ",a%
```

Es erscheint ein Alert-Requester mit zwei Gadgets. Nach der Wahl eines Gadgets erscheint ein zweiter Requester, der angibt, welches Gadget im ersten Requester gewählt wurde. Dieser zweite Alert-Requester besitzt keinen Default-Button.

FILESELECT titel$,oktext$,pfad$,name$

titel$,oktext$,pfad$: sexp
name$: svar

Dieser Befehl läßt die Datei-Auswahl-Box auf dem Bildschirm erscheinen, mit dessen Hilfe aus dem Inhaltsverzeichnis einer Diskette (RAM-Disk, Festplatte, usw.) eine Datei ausgewählt werden kann. „titel$“ ist der Aktionsname, der in der Titelzeile erscheinen soll. In „oktext$“ wird der Name des Ok-Gadgets (links) angegeben. Durch die Laufwerk-Gadgets, deren Anzahl sich nach der Anzahl der angeschlossenen Laufwerke richtet, ist es sehr einfach, die Laufwerke zu wechseln.

Im Ausdruck „pfad$“ steht der Name des Laufwerks und des Ordners, dessen Inhaltsverzeichnis aufgelistet werden soll. In „pfad$“ kann auch ein eventuell vorhandener Dateiname angegeben werden, wenn es nur darum geht, die Auswahl bestätigen zu müssen. Ist der Verzeichnispfad „pfad$“ ein Leerstring, so wird das aktuelle Laufwerk gewählt.

Werden irgendwelche Eingaben im Verzeichnispfad vorgenommen und mit Enter oder Return bestätigt, wird entsprechend des neuen Pfades gesucht. Ist der neu eingegebene Pfad nicht gefunden worden, wird das Anzeigefeld der Auswahl-Box leer angezeigt.

In „name$“ befindet sich der Name der gewählten Datei, nachdem man das Ok-Gadget der Auswahl-Box angeklickt hat. Wurde das Abbruch-Gadget gewählt, so enthält name$ einen Leer-String.

Das Format von name$ entspricht den Konventionen des hierarchischen Dateisystems, das im Kapitel „Allgemeine Ein- und Ausgabe“, Abschnitt „Dateiverwaltung“, beschrieben wird.

Beispiel:

```
FILESELECT "DF0:","Datei laden","Laden",name$
  IF name$=""
    PRINT "Sie haben den Abbruch–Knopf gewählt."
  ELSE IF RIGHT$(name$)=":"
    PRINT "Sie haben den OK–Knopf ohne Dateiangabe gewählt."
  ELSE
    PRINT "Sie haben die Datei: ";name$;" gewählt."
ENDIF
```

Läßt eine Datei-Auswahl-Box erscheinen und analysiert die vom Benutzer getroffene Wahl.

10 – Fortgeschrittene Programmierung

_dosCmd$
run gfabasic [[–] programm.gfa [kommando]]

Sie können GFA-BASIC nicht durch Anklicken des Icons starten, sondern auch durch den Aufruf mit CLI. Dabei können Sie den Namen eines BASIC-Programmes angeben, das direkt nach dem Laden von GFA-BASIC automatisch ausgeführt wird. Soll das BASIC-Programm nur in den Editor geladen, aber nicht gestartet werden, müßen Sie vor dem BASIC-Programmnamen ein Minuszeichen angeben (bitte Leerzeichen nicht vergessen!).

Diese Form des Aufrufs kann auch benutzt werden, um BASIC-Programme in Befehlsdateien (Batch-Datei) zu benutzen. In diesem Fall ist es möglich nach dem Namen des BASIC-Programmes noch weitere Zeichen anzugeben. Diese stehen in GFA-BASIC in der Variable _dosCmd$ zur Verfügung. Auch von der Workbench aus können Parameter, zumindest Dateinamen, übergeben werden. Dazu wird bei gedrückter Shift-Taste das Icon für GFA-BASIC-Programm selektieret und danach das Icon für die Parameter-Dateien. Dies wird dann durch Doppelklick abgeschlossen.

EXEC command$,in,out

command$:sexp
in, out: iexp

Dieser Befehl startet einen neuen CLI-Prozeß. Dazu geben Sie in „command$" ein CLI-Kommando an. Die Parameter „in" und „out" sind Kanalnummern der Standardein- und -ausgabekanäle für diesen CLI-Prozeß. Dabei kann der Wert –1 angegeben werden, wenn kein Kanal vorhanden ist.

Einige Beispiele:

```
EXEC ":clock",-1,-1
```

Startet die Uhr (sofern sich das Programm clock im Hauptverzeichnis des aktuellen Laufwerks befindet). GFA-BASIC wartet bis die Uhr gestoppt (das Fenster geschlossen) wird.

```
EXEC "run clock ANALOG=100,100,200,120 seconds"
```

Startet die paralell zum GFA-BASIC laufend.

Wurde GFA-BASIC selbst von CLI gestartet, wird dieses CLI-Fenster als Eingabe-/Ausgabekanal für den mit EXEC gestarteten Prozeß benutzt, sofern Sie —1 als Kanalnummer angeben. Wurde GFA-BASIC von Workbench gestartet, verwendet EXEC ein zusätzliches Fenster (CON:0/0/320/200/NewCLI Window) als Ausgabe.

Library-Variablen

_DosBase
_IntBase
_GfxBase
_LayersBase

Diese Variablen liefern die Adresse der jeweiligen Library-Basis. Diese Libraries werden von GFA-BASIC intern verwendet und sind aus diesem Grund im Hauptspeicher vorhanden. Sie können Funktionen dieser Libraries direkt aufrufen (siehe Kapitel 11), ohne die Library vorher öfnen zu müßen.

GFA-BASIC verwendet z.Zt. folgende Libraries:

_DosBase	DOS-Library (Die AmigaDos-Funktionen).
_IntBase	Intuition-Library (Die Oberfläche Intuition).
_GfxBase	Graphics-Library (Grafik- und Zeichenfunktionen).
_LayersBase	Layers-Library (Verwaltung der Zeichenebenen).

SPRITE #n,data$
SPRITE #n [,""]
SPRITE #n,x,y
SPRITE ON
SPRITE OFF

n, x, y: iexp
data$: sexp

Dieser Befehl stellt ein Hardware-Sprite auf dem Bildschirm dar. Das Sprite hat dabei eine Breite von 16 Punkten und eine beliebige Höhe. Die Auflösung des Sprites entspricht der niedrigen Bildschirmauflösung (320x200). Jeweils zwei Bits der Sprite-Definition ergeben einen Bildschirmpunkt und definieren dessen Farbe.

Mit dem Sprite-Befehl können die Hardware-Sprites des Amiga benutzt werden. Es gibt acht Sprites mit den Nummern 0 bis 7, wobei Sprite #0 durch die Maus belegt ist.

SPRITE #n,data$ definiert das Aussehen eines Sprites. In data$ steht dabei das Bitmuster für die Sprite-Datas. Jeweils 4 Byte sind für eine Zeile zuständig, die ersten zwei Byte für die Bitebene 0, die anderen für die Bitebene 1. Die Höhe der Sprites ergibt durch die Länge des Strings geteilt durch 4.

SPRITE #n gibt den entsprechenden Sprite frei, der durch Sprite #n,data$ angefordert wurde. Das gleiche gilt für SPRITE #n,"".

SPRITE #n,x,y bewegt Sprite n an die Position mit den Koordinaten x,y. Diese Koordinaten sind immer in der niedrigen Auflösung angegeben (320 * 256 oder 320 * 200). Vor dem Bewegen eines Sprites muß ein Fenster oder Screen geöffnet sein.

Mit SPRITE OFF werden alle Sprites abgeschaltet, auch der Mauszeiger. Mit SPRITE ON werden entsprechend alle definierten Sprites eingeschaltet.

Jeweils zwei Sprites (0/1, 2/3, usw.) benutzen diesselben Farbregister zur Definition der Spritefarben. Diese Farbregister sind für jeden Screen seperat festgelegt. Die Sprites benutzen immer die Farbregister des Screens, der im Hintergrund der Sprites liegt. Die Farbregister des Screens können mit SETCOLOR verändert werden.

Farbregister-Zuordnung:

Farbbits	Sprite 0/1	Sprite 2/3	Sprite 4/5	Sprite 6/7
00		immer transparent		
01	Reg. 17	Reg. 21	Reg. 25	Reg. 29
10	Reg. 18	Reg. 22	Reg. 24	Reg.30
11	Reg. 19	Reg. 23	Reg. 25	Reg. 31

Beispiel:

```
OPENW 1
a$=STRING$(64,-1)
SPRITE #2,a$
SETCOLOR 23,&hF00
FOR i%=0 TO 300
  SPRITE #2,i%,i%
  PAUSE 1
NEXT i%
SPRITE #2
```

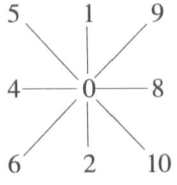
Dieses Programm bewegt ein hellrotes viereckiges Sprite diagonal über den Bildschirm.

STICK(p)
STRIG(p)

m,p: iexp

Der Amiga verfügt über zwei Schnittstellen (Ports) zum Anschluß von Maus und Joystick. Am Port 0 können Maus- oder Joystick-Koordinaten abgefragt werden, am Port 1 nur Joystick-Daten.

Die Funktion STICK(p) dient der Abfrage der Joystick-Position. Für p = 0 wird die Position des Joysticks in Port 0, für p = 1 die des Joysticks in Port 1 gemeldet.



Entsprechend ermittelt STRIG(p) den Staus des Feuerknopfes (TRUE = gedrückt, sonst FALSE).

Die Abfrage des Joysticks in Port 0 erfolgt am Intuition vorbei, d.h. das Betriebssystem reagiert auf Joystick-Bewegungen wie auf die Maus.

Aus Kompatiblilitätsgründen ist der Befehl STICK ebenfalls implementiert, hat aber keine Wirkung.

Beispiel:

```
                !Joystick in Port 1 abfragen,
REPEAT
  richtung%=STICK(1)
  feuer!=STRIG(1)
  SELECT richtung%
  CASE 4
    PRINT "Links"
  CASE 8
    PRINT "rechts"
  CASE 2
    PRINT "unten"
  CASE 1
    PRINT "oben"
  ENDSELECT
UNTIL feuer!
WHILE STRIG(1)
WEND                    ! Wartet auf das Loslassen des Feuerknopfes
```

SAY x$[,m%()]
TRANSLATE$()

x$ sexp
m%(): Integerfeld

Mit dem SAY-Befehl wird eine Zeichenkette im Phonem-Code vom Amiga „ausgesprochen". Meist wird dieser Befehl verwendet in Verbindung mit TRANSLATE $. Die Funktion TRANSLATE$ wandelt englischen Klartext in Phonem-Code um. Für deutsche Texte empfiehlt sich oft bewußt falsch zu schreiben.

Damit SAY funktioniert, muß sich im DEVS:-Ordner das narrator.device befinden. Für TRANSLATE$ ist im LIBS:-Ordner die translator.library erforderlich.

Mit dem Feld m%() kann die Ausgabe von SAY modifiziert werden:

m%(0)	Grundfrequenz (65-320; Default=110)
m%(1)	Modulation (0=natürlich, 1=monoton; Default=0)
m%(2)	Sprechgeschwindigkeit (40-400 Worte/min; Default=150)
m%(3)	Geschlecht (0=männlich, 1=weiblich; Default=0)
m%(4)	Sample-Frequenz (5000-28000, Default=22200)
m%(5)	Lautstärke (0-64; Default=64)
m%(6)	Tonkanal (Wert=Kanal: 0=0, 1=1, 2=2, 3=3, 4=0 +1, 5=0+2 6=3+1, 7=3+2, 8=0 oder 3, 9=1 oder 2, 10=0 oder 3 und 1 oder 2 11=0, 1, 2 oder 3; Default=10 (Stereo))

Beispiel:

```
SAY TRANSLATE$("I am an Amiga")
```

Mit dem folgenden Programm werden aus den dem Amiga beiliegenden FD-Files GFA-Prozeduren als .LST-Files erzeugt, die dann mit Merge zu dem Programmen hinzugeladen werden können. Für DOS, Intuition, Graphics, Layers usw. (s. Kapitel 11) ist dies in der Regel unnötig.

```
DIM a$(15)
OPEN "i",#1,":fd1.3/dos_lib.fd"
DO UNTIL EOF(#1)
  LINE INPUT #1,a$
  IF LEFT$(a$)="#"
    SELECT CVL(MID$(a$,3))
    CASE "base"
      SELECT UPPER$(MID$(a$,8))
      CASE "_DOS"
        l$="_DosBase"
      CASE "_LAY"
        l$="_LayersBase"
      CASE "_INT"
        l$="_IntBase"
      CASE "_GFX"
        l$="_GfxBase"
      CASE "_SYS"
        l$="{4}"
      DEFAULT
        l$=MID$(a$,9)+"%"
      ENDSELECT
    CASE "bias"
      o%=VAL(MID$(a$,8))
    CASE "publ"
    CASE "priv"
    ENDSELECT
  ELSE IF LEFT$(a$)"*"
    PRINT "> PROCEDURE ";
    n$=LEFT$(a$,INSTR(a$,")"))
    nl$=LEFT$(n$,INSTR(a$,"(")-1)
    PRINT nl$;
    nr$=MID$(n$,LEN(nl$)+2)
    i%=0
    WHILE LEN(nr$)1
      a%=INSTR(nr$,",")
      IF a%=0
        a%=LEN(nr$)
      ENDIF
      a$(i%)=LEFT$(nr$,a%-1)
```

```
      INC i%
      nr$=MID$(nr$,a%+1)
    WEND
    IF i%                !wenn mit parametern
      a%=1
      WHILE a%%
        b%=0
        WHILE b%a%
          IF a$(a%)=a$(b%)
            a$(a%)=a$(a%)+STR$(a%)
            a$(b%)=a$(b%)+STR$(b%)
          ENDIF
          INC b%
        WEND
        INC a%
      WEND
      PRINT "(";
      j%=0
      WHILE j%%-1
        PRINT a$(j%);"%,";
        INC j%
      WEND
      PRINT a$(j%);"%)";
    ENDIF
    PRINT
    r$=MID$(a$,LEN(n$)+2)
    i%=0
    WHILE LEN(r$)
      SELECT ASC(r$)
      CASE "D"
        PRINT "   m68%(";MID$(r$,2,1);")=";a$(i%);"%"
      CASE "A"
        PRINT "   m68%(";VAL(MID$(r$,2,1))+8;")=";a$(i%);"%"
      ENDSELECT
      INC i%
      r$=MID$(r$,4)
    WEND
    PRINT "   m68%(14)=";l$
    PRINT "   RCALL ";l$;"-";o%;",m68%()"
    ADD o%,6
    PRINT "RETURN"
  ENDIF
LOOP
```

Zum Beispiel entsteht aus

```
##base_TranslatorBase
##bias 30
##public
Translate(inString,inLength,outBuffer,bufferSize)(A0,D0/A1,D1)
##end
```

```
 PROCEDURE Translate(inString%,inLength%,outBuffer%,bufferSize%)
   m68%(8)=inString%
   m68%(0)=inLength%
   m68%(9)=outBuffer%
   m68%(1)=bufferSize%
   m68%(14)=TranslatorBase%
   RCALL TranslatorBase%-30,m68%()
RETURN
```

Zur Anwendung dieser Routine fehlt nur noch DIM m68%(15) und das Open-Library.

Die Version 3.0 des GFA-BASIC Amiga ermöglicht das Erzeugen und Benutzen von BOBs (Blitterobjekten). Ein BOB ist eine spezielle Form von Sprite, die entweder mit dem Amiga-Basic-Objekteditor oder mit dem auf dieser Diskette befindlichen Programm IFF_TO_BOB.GFA erzeugt werden kann.

Diese BOBs eignen sich sehr gut zur Gestaltung eigener schneller Animationen; Kenntnisse der Maschinensprache sind dabei nicht nötig. Alle BOB-Kommandos sind voll kompatibel zu den Amiga-Basic OBJECT-Kommandos mit Ausnahme der folgenden, nicht implementierten Kommandos:

COLLISION ON/OFF/STOP
=COLLISION(-1) ergibt nicht die Fensternummer.

Es folgt eine kurze Übersicht über die OBJECT-Kommandos (Object (oder Obj) meint hier immer die Objektnummer):

OBJECT.AX obj,value
OBJECT.AY obj,value

value: Wert der Beschleunigung in Pixeln/sec.

Bestimmt die Beschleunigung eines Objekts für die x- und y-Richtung.

OBJECT.CLIP x0,y0,x1,y1

Legt das Clipping-Rechteck fest, innerhalb dessen die BOBs gezeichnet werden. Das Ändern der Fenstergröße hat keinen Einfluß auf die Clippingwerte.

OBJECT.CLOSE [Obj [,Obj [,Obj]]]

Löschen eines Objektes. Falls keine Parameter angegeben werden, löscht GFA-BASIC 3.0 Amiga alle aktiven BOBs.

OBJECT.ON [Obj [,Obj [,Obj]]]
OBJECT.OFF [Obj [,Obj [,Obj]]]

Hiermit werden BOBs ein- oder ausgeschaltet; dabei ist Obj die Nummer des betreffenden Objekts mit 0 <= Obj < 16. Werden keine Parameter aufgeführt, dann gilt das Kommando für jeweils alle BOBs.

OBJECT.PLANES Obj [,PlanePick [,PlaneOnOff]]

PlanePick, PlaneOnOff: 8-Bit-Masken

Hiermit wird die BOB-Farbe definiert. PlanePick bestimmt hierbei, in welche Bitplane die BOB-Information gezeichnet wird. PlaneOnOff legt die Verwendung für die von PlanePick nicht genutzten Bitplanes durch das Löschen oder Setzen von Bitplanes fest (siehe hierzu auch das Amiga-Basic Handbuch).

OBJECT.PRIORITY obj,prio

Festlegen der Prioritätsfolge der BOBs. Hierbei werden die BOBs in der Reihenfolge der hier vergebenen Priorität übereinander gezeichnet.

0 <= Prio <= 32768

Werden keine Prioritäten festgelegt, dann werden die BOBs in einer willkürlichen Reihenfolge übereinander gezeichnet.

OBJECT.SHAPE obj,string
OBJECT.SHAPE obj1,obj2

Hier gibt es zwei mögliche Formen :

1) obj ist die Nummer des zu definierenden BOBs, string enthält die BOB-Definition aus dem Amiga-Basic Objekt-Editor oder aus dem IFF_TO_BOB.GFA Programm.

2) obj1 ist die Nummer eines existierenden BOBs, dessen exakte Kopie unter der Nummer obj2 angelegt wird mit Ausnahme der BOB-Position, des Kollisionshandling und der Farbmaske.

OBJECT.START [Obj [,...]]
OBJECT.STOP [Obj [,...]]

Starten und Stoppen der BOB-Bewegung. Werden keine Objektnummern aufgeführt, gelten die Kommandos für alle BOBs.

OBJECT.VX obj,value
OBJECT.VY obj,value

value: Wert der Geschwindigkeit in Pixeln/sec.

Bestimmt die Geschwindigkeit eines Objekts für die x- und y-Richtung.

OBJECT.X obj,value
OBJECT.Y obj,value

Obj : Objektnummer
value : Pixelwerte der Koordinaten

Festlegen der Position (x,y) des BOBs im Fenster.

Um den BOB sichtbar zu machen, muß das OBJECT.ON Kommando benutzt werden. Die BOB-Bewegung wird mit OBJECT.START eingeleitet, nachdem die Geschwindigkeit und die Beschleunigung des BOBs definiert worden sind.

=OBJECT.X(obj)
=OBJECT.Y(obj)
=OBJECT.VX(obj)
=OBJECT.VY(obj)
=OBJECT.AX(obj)
=OBJECT.AY(obj)

Mit diesen Funktionen können für gegebene Objekte Position, Geschwindigkeit und Beschleunigung abgefragt werden.

ON COLLISION GOSUB proc

proc : Name der Prozedur, die bei Kollisionen aufgerufen wird.

11 – Systemroutinen

Aufruf der Systemroutinen

In GFA-BASIC besteht die Möglichkeit Funktionen des Betriebssystems auszuführen. Bei der Eingabe solcher Funktionen wird automatisch die Syntax und die Schreibweise kontrolliert und gegebenenfalls korrigiert. Da die meisten Funktionen mit mehreren Parametern angegeben werden müssen, sollte man sich der Korrektheit der Angaben sicher sein.

Der Interpreter fängt solche unkorrekten Angaben nicht ab, sondern überläßt dem System die Entscheidung der entsprechenden Fehlermeldung (GURU).

Werden Funktionen eingesetzt, die in ähnlicher Form auch als BASIC-Befehle existieren (OPENW und OpenWindow), sollten auch alle weiteren Zugriffe auf deren Meldungen und Resultate mit Betriebssystems-Funktionen durchgeführt werden.

Die Beschreibung zu allen Funktionen beschränkt sich auf das Wesenlichste. Zu jeder Funktion erfolgt eine kurze Erläuterung und die Beschreibung der Parameter und des Rückgabewertes.

AbortIO(IO_request)

Io_request: iexp

Versucht den letzten I/O-Request abzubrechen. Keine Rückgabe.

Io_request: Adresse einer I/O-Request-Struktur.

AddDevice(adr)

adr:iexp

Fügt ein neues Device zur System-Device-Liste und macht es für alle Tasks zugängig. Keine Rückgabe.

adr: Adresse der initialisierten Device-Struktur.

AddFreeList(freeList,memBlock,length)

freeList, memBlock, length: iexp

Diese Funktion fügt einen bestimmten Speicherbereich an eine Free-Memory Liste. Rückgabe: im Fehlerfall ist es 0. Der Fehler kann mit IoErr() abgefragt werden.

freeList: Adresse der FreeList Struktur.
memBlock: Adresse des anzufügenden Blockbereichs.
length: Speichergröße in Bytes.

AddHead(list,node)

list, node: iexp

Fügt eine neue Node-Struktur an den Anfang einer doppelt verketteten Liste. Keine Rückgabe.

list: Adresse der List-Struktur.
node: Adresse der Node-Struktur.

AddIntServer(n,adr)

n, adr: iexp

Fügt eine neue Interrupt-Server Routine zu den bereits bestehenden. Keine Rückgabe.

n: PAULA (peripherals-sound custom chip) Interrupt Bit.
adr: Adresse der Interrupt Struktur.

AddLibrary(adr)

adr:iexp

Fügt ein neues Library zum System und macht es für alle Tasks zugängig. Keine Rückgabe.

adr: Adresse der initialisierte Library Struktur.

AddPort(msgPort)

msgPort: iexp

Mit dieser Funktion wird ein neuer Message Port an die System-Message-Port-Liste angefügt und jedem Task verfügbar gemacht. Keine Rückgabe.

msgPort: Adresse der MsgPort-Struktur.

AddResource(resource)

resource: iexp

Mit dieser Funktion wird ein Resource an die System-Resource-Liste angefügt und jedem Task verfügbar gemacht. Keine Rückgabe.

resource: Adresse der initialisierte Resource Struktur.

AddTail(list,node)

list, node: iexp

Fügt eine neue Node-Struktur an das Ende einer doppelt Verketteten Liste. Keine Rückgabe.

list: Adresse der List-Struktur.
node: Adresse der Node-Struktur.

AddTask(taskCB,initialPC,finalPC)

taskCB, initialPC, finalPC: iexp

Dient zum Anmelden einer neuen Task. Keine Rückgabe.

taskCB: Adresse der Control-Block-Struktur der Task.
initialPC: Adresse des Stack-Anfangs.
finalPC: Adresse des Stack-Endes.

Allocate(memheader,byteSize)

memheader, byteSize: iexp

Allocate fordert einen Block aus einer eigenen Speicherlistenverwaltung an. Rückgabe: Konnte die Funktion erfolgreich ausgeführt werden, erhält man die Adresse des Blockanfangs andernfalls 0.

memheader: Adresse der MemHeader-Struktur.
byteSize: Blockgröße in Bytes.

AllocEntry(memList)

memList: iexp

Mit dieser Funktion wird eine MemList-Struktur initialisiert. Rückgabe: Konnte die Funktion erfolgreich ausgeführt werden, erhält man die Adresse der MemList-Struktur. Trat ein Fehler auf, ist das höchste Bit des 32 Bit-Wertes gesetzt.

memList: Adresse der MemList-Struktur.

AllocMem(byteSize,art)

byteSize, art: iexp

Diese Funktion reserviert Speicher (MALLOC). Rückgabe: Adresse des zusammenhängenden Blocks oder 0, falls nicht verfügbar.

byteSize: Größe des Blocks in Bytes. Der Block wird immer auf die nächste 8 Byte-Grenze gerundet.
art: Speicherattribute. (z.B. MEMF_CHIP)

AllocSignal(signalNum)

signalNum: iexp

Dient zur Reservierung eines Signalbits von allen verfügbaren Signalbits der laufenden Task.

Rückgabe: Bitnummer oder –1.
signalNum: Bitnummer 0–31 oder –1 für kein bestimmtes Bit.

AllocTrap(trapNum)

trapNum: iexp

Reserviert eine Trapnummer der verfügbaren Traps der laufenden Task. Rückgabe: Trapnummer oder –1.

trapNum: Trapnummer 0 – 15 oder –1 für keinen bestimmten Trap.

AvailMem(art)

requirements: iexp

Liefert den freien Speicher. Rückgabe: freier Speicher.

art: Speicherattribute.

Cause(interrupt)

interrupt: iexp

Führt einen Software Interrupt aus. Keine Rückgabe.

interrupt: Adresse der initialisierten Interrupt-Struktur.

CloseDevice(Io_request)

ioRequest: iexp

Teilt dem System mit, daß ein Device abgemeldet wird. Keine Rückgabe.

Io_request: Adresse einer initialisierten I/O-Request-Struktur.

CloseLibrary(library)

library: iexp

Schließt eine Funktions-Library. Keine Rückgabe.

library: Adresse der Library-Struktur.

Deallocate(memHeader,memBlock,byteSize)

memHeader, memBlock, byteSize: iexp

Diese Funktion gibt einen Block aus einer eigenen Speicherverwaltungsliste frei. Keine Rückgabe.

memHeader: Adresse der MemHeader-Struktur.
memBlock: Adresse des freizugebenden Speicherbereichs.
byteSize: Größe des Blocks in Bytes.

CreateExtIO(replyport,größe)

replyport, größe: iexp

Diese Funktion reserviert Speicher und initialisiert eine erweiterte IO-Request-Struktur. Rückgabe: Die Adresse der IO-Request-Struktur.

replyport:	Adresse der bereits reservierten und initialisierten Message.
größe:	Größe des zu reservierenden Speichers. Diese muß mindestens so groß wie die Struktur sein, für die der Speicher angefordert werden soll.

CreateProc(name,pri,seg,stack)

name: sexp
seg: iexp
pri, stack: ivar

Startet einen neuen Prozess. Rückgabe: Prozeß-ID

name:	Namen-String für den Prozeß.
pri:	Prozeß-Prioriztät (–128 bis 127).
seg:	Adresse des Segments (SegLoad()).
stack:	Größe des Stacks für den Prozeß.

CreateStdIO(replyport)

replyport: iexp

Reserviert Speicher und initialisiert eine neue IOStdReq-Struktur. Rückgabe: Adresse der Struktur.

replyport:	Adresse eines bereits initialisierten Message-Ports

Debug()

Debug aktiviert die ROM-Wack Funktion des Betriebssystems. Danach werden die Debug-Daten über die RS-232 Schnittstelle (9600 Baud) geschickt, um von anderen Systemem empfangen zu werden.

DeviceProc(name)

name: iexp

DeviceProc ermittelt die Adresse des in „name“ spezifizierten Device-Handlers. Rückgabe: Adresse des Prozesses.

name: Prozeß-Name

Disable()

Disable ist der Funktion „Forbid“ sehr ähnlich. Hierbei werden allerdings noch zusätzlich Interrupts gesperrt.

DoIO(Io_request)

Io_request: iexp

Diese Funktion versucht, das in der I/O-Request-Struktur festgelegte Kommando auszuführen.

Rückgabe: Bei Erfolg 0, im Fehlerfall die Fehlernummer.
Io_request: Adresse der initialisierten I/O-Request-Struktur.

Enable()

Aktiviert Interrupts, nachdem sie durch die Funktion „Disable“ deaktiviert wurden.

Enqueue(list,node)

list, node: iexp

Enqueue fügt eine Node-Stuktur in die spezifizierte System-Liste. Keine Rückgabe.

list: Adresse der List-Struktur.
node: Adresse der Node-Struktur.

FindName(start,name)

start, name: iexp

FindName durchsucht eine System-Node-Liste nach der in „name“ angegeben Namen der Node Stuktur.

Rückgabe: Adresse der Node-Struktur oder 0.
start: Adresse einer Node- oder List-Struktur.
name: Adresse des Namen-Strings der Node-Struktur.

FindPort(name)

name: iexp

Sucht in der System-Message-Port-Liste nach dem Port mit dem angegebenen Namen. Rückgabe: Adresse der MessagePort-Struktur oder 0.

name: Adresse des Namen-Strings der MessagePort-Struktur.

FindResident(name)

name: iexp

Diese Funktion versucht die Adresse einer Resident-Struktur zu finden. Rückgabe: Adresse der Resident-Struktur oder 0.

name: Adresse des Namen-Strings der Resident-Struktur.

FindTask(name)

name: iexp

Durchsucht die Task-Listen nach dem Task mit dem angegebenen Namen. Rückgabe: Adresse der TaskControlBlock-Struktur oder, falls ein leerer String übergeben wurde, die Adresse des aktuellen Tasks.

name: Adresse des Namen-Strings der Task-Struktur.

Forbid()

Forbid schaltet das Taskswitching ab.

FreeEntry(memlist)

memlist: iexp

FreeEntry löscht alle Speicherbereiche, die durch die Funktion AllocEntry() in einer MemList-Struktur reserviert wurden. Keine Rückgabe.

memlist: Adresse der MemList-Struktur, die die MemEntry-Strukturen enthält.

FreeFreeList(freelist)

freelist: iexp

Löscht alle Einträge in der angegebenen FreeList-Struktur. Keine Rückgabe.

freelist: Adresse der FreeList-Struktur.

FreeMem(adr,count)

adr: iexp
count: ivar

Löscht einen Speicherbereich, der mit der Funktion AllocMem() reserviert wurde. Rückgabe: gelöschte Blockgröße in Bytes.

adr: Adresse des Speicherbereichs.
count: Größe des Speicherbereichs in Bytes.

FreeRaster(memblock,w,h)

memblock: iexp
w, h: ivar

Diese Funktion löscht eine Bitplane einer Rasterbitmap, die mit der Funktion AllocRaster() reserviert wurde. Keine Rückgabe.

memblock: Adresse des Speicherbereichs.
w, h: Weite und Höhe der Bitplane.

FreeSignal(sigbit)

sigbit: iexp

Gibt ein mit der Funktion AllocSignal reserviertes Signal-Bit wieder frei. Keine Rückgabe.

sigbit: Das freizugebende Signal-Bit (0–31).

FreeTrap(trapnr)

trapnr: iexp

Gibt einen mit der Funktion AllocTrap reservierten Trap wieder frei. Keine Rückgabe.

trapnr: Nummer des Traps (0–15).

GetCC()

Diese Funktion ließt den Zustands-Code der 68000-CPU aus. Rückgabe: Code-Bits (0–4)

GetMsg(Msgport)

msgport: iexp

Die Funktion GetMsg holt die nächte Nachricht aus dem angegebenen Message-Port. Nachdem die Nachricht empfangen wurde, muß sie mit der Funktion ReplyMsg() wieder zurückgeschickt werden, um zu signalisieren, daß sie empfangen wurde. Rückgabe: Adresse des Nachrichten-Ports, in den die Nachricht kopiert wird.

MsgPort: Adresse der MsgPort-Stuktur von der Nachrichten empfangen werden sollen.

InitCode(flag,version)

flag, version: iexp

InitCode initialisiert alle Resident-Module mit einem angegebenen Startwert und einer Versionsnummer. Keine Rückgabe.

flag: Startwert.
version: Versionsnummer des Resident-Moduls.

InitResident(resident,seglist)

resident, seglist: iexp

InitResident initialisiert eine Resident-Struktur unter Benutzung einer Segment-Liste, die die Initialisierungswerte enthält. Keine Rückgabe.

resident: Adresse einer Resident-Struktur
seglist: Adresse der Segment-Liste für die Resident-Struktur.

InitStruct(init,buffer,größe)

init, buffer: iexp
größe: ivar

Mit dieser Funktion kann ein Speicherbereich intialisiert werden. Zur Initialisierung des Speichers werden die Werte, die ab Adresse „init" im RAM stehen, verwendet. Keine Rückgabe.

init: Adresse der Werte-Tabellen für die Initialisierung.
buffer: Adresse des Speicherbereichs, der initialisiert werden soll.
größe: Größe des Speicherbereichs in Byte.

Insert(list,node1,node2)

list, node1, node2: iexp

Diese Funktion fügt eine Node-Struktur in eine Node-Strukturen-Liste. Keine Rückgabe.

list:	Adresse einer List-Struktur, in der die Node-Struktur enthalten ist.
node1:	Adresse der Node-Struktur, die eingefügt werden soll.
node2:	Adresse der Node-Struktur, hinter die „node1“ eingefügt werden soll.

MakeLibrary(funcinit,stMakeLibrary(funcinit,structinit,libinit, größe,seglist)

funcinit, structinit, libinit, seglist: iexp
größe: ivar

Diese Funktion erzeugt eine neue Funktions-Bibliothek. Rückgabe: Adresse der Library-Struktur.

funcinit:	Adresse der Funktions-Adressenliste.
structinit:	Adresse des Speicherbereichs, der mit der Funktion InitStruct gefüllt werden soll.
libinit:	Adresse der INIT-Funktion, die aufgerufen werden muß, bevor die Funktions-Bibliothek dem System zugänglich gemacht werden kann.
größe:	Die Größe des Speicherbereichs der Funktionen und der Library-Struktur.
seglist:	Adresse einer AmigaDOS-Speicher-Segment-Liste. Diese wird nur für AmigaDos-Librarys benötigt.

OpenDevice(devname,nummer,iorequest,flags)

devname, nummer, iorequest, flags: iexp

Diese Funktion öffnet das angegebene Device und initialisiert die IORequest-Struktur. Rückgabe: Bei Erfolg 0, andernfalls eine Fehlernummer.

devname:	Der Name des zu öffnenden Devices.
nummer:	Die Devicenummer (0–4).
iorequest:	Adresse einer IORequest-Struktur, die dabei initialisiert wird.
flags:	Enthält spezielle Device-Informationen.

OpenLibrary(name,version)

name: sexp
version: iexp

Diese Funktion öffnet eine Funktions-Bibliothek und macht diese dem System zugänglich. Rückgabe: Adresse der Library-Struktur.

version: Versionsnummer der Funktion-Bibliothek.

OpenResource(name)

name: sexp

Diese Funktion öffnet eine bereits installierte Hilfsdatei und liefert die Adresse einer Resource-Struktur zurück. Rückgabe: Adresse einer Resource-Struktur oder 0.

name: Name der Resource-Datei.

Permit()

Schaltet das Taskswitching wieder ein, nachdem es mit der Funktion Forbid() ausgeschaltet wurde.

Procure(semphore,message)

semophore, message: iexp

Wird die angegebene Semaphore-Struktur gerade nicht genutzt, liefert die Funktion TRUE zurück und die Message-Struktur enthält keine Nachricht. Ist sie allerdings doch in Gebrauch, wird FALSE zurückgemeldet. Rückgabe: TRUE oder FALSE

semaphore: Adresse der Semphore-Struktur.
message: Adresse der Message-Struktur.

PutMsg(msgport,message)

msgport, message: iexp

Diese Funktion hängt eine Nachricht an den angegebenen Message-Port. Keine Rückgabe.

msgport:	Adresse der MessagePort-Struktur, die die Nachricht empfangen soll.
message:	Adresse der Message-Struktur, deren Inhalt übermittelt werden soll.

RawDoFmt(string,streamadr,function,buffer)

string, streamadr, function, buffer: iexp

Mit dieser Funktion können Daten formatiert werden. Keine Rückgabe.

string:	Adresse eines mit Null-Byte endenden Strings.
streamadr:	Adresse des zu formatierenden Speicherbereichs.
function:	Adresse der Ausgabefunktion.
buffer:	Adresse eines Speicherbereichs, in den die formatierten Daten geschrieben werden sollen.

RawKeyConvert(event,buffer,len,keymap)

event, buffer, len, keymap: iexp

Diese Funktion wandelt den Scancode einer Taste anhand der angegebenen Keymap in das entsprechende ASCII-Zeichen um. Rückgabe: Es wird die Anzahl der Zeichen im Puffer zurückgemeldet.

event:	Adresse der Input-Event-Struktur.
buffer:	Byte-Puffer.
len:	Größe des Puffers in Bytes.
keymap:	Adresse der zu verwendenden Keymap.

RemDevice(device)

device: iexp

Löscht ein existierendes Device aus der System-Device-Liste. Keine Rückgabe.

device: Adresse einer Device-Struktur.

RemHead(list)

list: iexp

RemHead löscht eine Node- Struktur aus einer Liste von Node-Strukturen. Rückgabe: Adresse der gelöschten Node-Struktur.

list: Adresse der List-Struktur.

RemLibrary(library)

library: iexp

Löscht eine Funktions-Bibliothek aus der Library-Liste des Systems. Danach ist sie nicht mehr zugänglich, selbst wenn erneut die Funktion AddLibrary() aufgerufen werden sollte. Rückgabe: 0, wenn kein Fehler aufgetreten ist. Anderenfalls eine Fehlernummer.

library: Adresse einer Library-Struktur.

Remove(node)

node: iexp

Löscht eine Node-Struktur aus einer verketteten Liste von Node-Strukturen. Keine Rückgabe.

node: Adresse der Node-Struktur, die gelöscht werden soll.

RemPort(msgport)

msgport: iexp

Ein allgemeiner Message Port wird aus der Message-Port-Liste gelöscht. Keine Rückgabe.

msgport: Adresse einer MsgPort-Struktur.

RemResource(resource)

resource: iexp

Diese Funktion löscht eine Hilfsdatei aus der Resource-Liste des Systems. Keine Rückgabe.

resource: Adresse der Resource-Struktur.

RemTail(list)

list: iexp

Diese Funktion liefert die Adresse der letzten Node-Struktur der Liste und löscht diese. Rückgabe: Adresse der Node-Struktur.

list: Adresse der List Struktur.

RemTask(task)

task: iexp

Diese Funktion beendet den angegebenen Task und löscht ihn. Keine Rückgabe.

task: Adresse der Task-Struktur.

ReplyMsg(message)

message: iexp

Immer wenn eine Nachricht in einem Message-Port anliegt, muß diese, nachdem sie analysiert wurde, wieder an den sendenden Task zurückgeschickt werden. Keine Rückgabe.

message: Adresse einer Message-Struktur.

SendIO(IO_request)

IO_request: iexp

Diese Funktion breitet einen „Device-Treiber" auf das in der IORequest-Struktur angegebene Kommando vor. Keine Rückgabe.

IO_request: Adresse einer initialisierten IORequest-Struktur.

SetExept(new,mask)

new, mask: iexp

Mit dieser Funktion kann festgelegt werden, welches der 32 Signal-Bits ausgewählt werden soll. Rückgabe: Das alte Signal-Bit.

new: Das neue Signal-Bit.
mask: Eine 32-Bit-Maske.

SetFunction(library,funkoffset,funkadr)

library, funkoffset, funkadr: iexp

Mit SetFunktion kann eine neue Funktion in eine Funktions-Bibliothek eingefügt werden. Rückgabe: Adresse der alten Funktion, die an der angegebenen Stelle in der Bibliothek stand.

libray: Adresse der Libray-Struktur, in die die neue Funktion aufgenommen werden soll.
funkoffset: Die Position innerhalb der Bibliothek.
funkadr: Die Adresse der Funktion, die eingefügt werden soll.

SetIntVector(nummr,interrupt)

nummr: ivar
interrupt: iexp

Setzte den System-Interrupt-Vector neu. Rückgabe: Adresse der alten Interrupt-Struktur.

nummer: Die Bit Nummer(0–14) der PAULA (peripherals-sound custom chip)
interrupt: Adresse der neuen Interrupt-Struktur.

SetSignal(newsig,sigmask)

newsig, sigmask: iexp

SetSignal setzt den Status der Task empfangs Signal-Bits. Rückgabe: Altes Signal-Bit.

newsig: Neues Signal-Bit.
sigmask: 32-Bit-Maske.

SetSr(neu,mask)

neu, mask: iexp

Mit dieser Funktion kann das Statusregister der 68000-CPU verändert werden. Rückgabe: Altes Statusregister

neu: Neuer Bitwert für das Statusregister.
mask: Gibt an, welche Bits verändert werden dürfen.

SetTaskPri(task,pri)

task: iexp
pri: ivar

SetTaskPri verändert die Priorität einer Task. Rückgabe: alte Priorität.

task: Adresse der TaskControlBlock-Struktur.
pri: Neue Priorität (–128 bis 127).

SetWrMsk(rastport,mask)

rastport, mask: iexp

Schützt bestimmte Bitplanes vor dem Überschreiben. Keine Rückgabe.

rastport: Adresse der RastPort-Struktur.
mask: Gibt an, welche Bitplanes vor dem Überschreiben geschützt werden sollen.

Signal(task,signal)

task, signal: iexp

Diese Funktion schickt ein Signal an eine Task, um beispielweise auf eine Nachricht hinzuweisen, wenn zuvor mit der Funktion PutMsg() eine Nachricht an die betreffende Task geschickt wurde. Keine Rückgabe.

task: Adresse der TaskControllock-Struktur.
signal: Signal-Bit

SumLibrary(library)

library: iexp

Errechnet eine neue Checksum für eine Library. Keine Rückgabe.

library: Adresse einer Library Struktur.

SuperState()

Diese Funktion schaltet das System in den Supervisior-Modus der 68000-CPU um.

SwapBitsRastPortClipRect(rastport,cliprect)

rastport, cliprect: iexp

Diese Funktion kopiert eine Rastport-Inhalt in die Bitmap eines Clip-Rechteckes. Keine Rückgabe.

rastport: Adresse der RastPort-Struktur.
cliprect: Adresse der ClipRect-Struktur.

UserState(stack)

stack: iexp

UserState schaltet das System in den User-Modus zurück. Keine Rückgabe.

stack: Die Größe des Supervisior-Stacks.

Vacate(semaphore)

semaphore: iexp

Umkehrfunktion zu Procur(). Keine Rückgabe.

semaphre: Adresse einer Semaphore-Struktur.

Wait(signal)

signal: iexp

Diese Funktion hält einen Task solange an, bis ein Signal geschickt wird.

signal: Signal-Bit

WaitIO(IO_request)

IO_request:iexp

Wartet bis das Kommando, das in der IORequest-Struktur angegeben wird, ausgeführt ist. Rückgabe: 0 oder eine Fehlernummer.

IO_request: Adresse einer IORequest-Struktur.

WaitPort(msgport)

msgport: iexp

Diese Funktion wartet auf ein Ereignis in dem angegebenen Message-Port.

msgport: Adresse der Message-Struktur.

AddVSprite(vSprite,rastport)

vSprite, rastport: iexp

Anfügen einer VSprite-Struktur an eine bestehende Graphik-Elemente-Liste. Keine Rückgabe.

vSprite: Adresse der VSprite-Struktur.
rastport: Adresse der kontrollierenden RastPort-Struktur.

AddAnimOb(animOb,animKey,rastport)

animOb, animKey, rastport: iexp

Fügt eine AnimOb-Struktur an eine Animation-Objekt Liste. Keine Rückgabe.

animOb: Adresse der anzufügenden AnimOb-Struktur.
animKey: Adresse der ersten AnimOb-Struktur in der Liste.
rastport: Adresse der RastPort-Struktur.

AddBop(bob,rastport)

bob, rastport: iexp

Diese Funktion dient zum Anhängen einer Bob-Struktur an die laufende Graphik-Element-Liste. Keine Rückgabe.

bob: Adresse der Bob-Struktur.
rastport: Adresse der RastPort-Struktur.

AddFont(textFont)

textFont: iexp

Fügt einen neuen Zeichensatz an die System-Zeichensatzliste und macht den Zeichensatz für alle Tasks zugänglich. Keine Rückgabe.

textFont: Adresse der TextFont-Strukur.

AllocRaster(width,height)

width, height: iexp

Allokiert Chip-Speicher für einzelne Ebenen von Bit-Planes. Rückgabe: Adresse des Blocks oder 0.

width: Breite der Bitplane.
height: Höhe der Bitplane.

AllocWBObject(wbobject)

wbobjet: iexp

Allokiert den nötigen Speicher für Worbench-.info-Dateien. Rückgabe: Adresse der WBObject-Struktur oder 0.

AndRectRegion(region,rectangle)

region, rectangle: iexp

Führt eine Und-Verknüpfung zwischen dem Clip-Rechteck und einem Grafikbereich durch. Das Resultat steht im Bereich, der durch die Region-Struktur festgelegt wird.

region: Adresse der Region-Struktur.
rectangle: Adresse der Rectangle-Struktur.

AndRegionRegion(region1,region2)

region1, region2: iexp

Führt eine Und-Verknüpfung zwischen zwei Grafikbereichen aus. Das Resultat steht in region2.

region1: Adresse der ersten Region-Struktur.
region2: Adresse der zweiten Region-Struktur.

Animate(animKey,rastPort)

animKey, rastPort: iexp

Diese Funktion bewegt selbsterstellte Animations-Objekte. Keine Rückgabe.

animKey: Adresse der ersten AnimOb Struktur.
rastPort: Adresse der RastPort-Struktur.

AreaCircle(rastPort,centerX,centerY,radius)

rastPort, centerX, centerY, radius: iexp

Fügt die Kreisinformationen an eine AreaInfo-Liste. Um die Area-Funktion ausführen zu können, muß mit den Funktionen InitTmpRas und InitArea ein Puffer-Speicher für den Blitter angefordert werden. Rückgabe: 0 oder im Fehlerfall 1.

rastPort: Adresse der RastPort-Struktur.
centerY: Y-Koordinate des Kreismittelpunkts.
radius: Radius.

AreaDraw(rastPort,x,y)

rastPort, x, y: iexp

Fügt einen Eckpunkt für ein Polygon an. Rückgabe: 0 oder im Fehlerfall –1.

rastPort: Adresse der RastPort-Struktur.
x: X-Koordinate des Punktes.
y: Y-Koordinate des Punktes.

AreaEllipse(rastPort,centerX,centerY,horiz_radius,vert_radius)

rastPort, centerX, centerY, horiz_radius, vert_radius: iexp

Fügt die Informationen der Ellipse an eine AreaInfo-Liste. Rückgabe: 0 oder im Fehlerfall 1.

rastPort: Adresse der RastPort-Struktur.
centerX: X-Koordinate des Mittelpunkts.

centerY: Y-Koordinate des Mittelpunkts.
horiz_radius: Horizontaler Radius.
vert_radius: Vertikaler Radius.

AreaEnd(rastPort)

Führt die AreaInfo-Liste des RastPorts aus. Dabei werden alle Objekte, die zuvor mit Area-Funktionen an die AreaInfo-Liste angehängt wurden, gezeichnet.

rastPort: Adresse der RastPort-Struktur.

AreaMove(rastPort,x,y)

rastPort, x, y: iexp

Legt einen Startpunkt für ein Polygon fest. Rückgabe: 0 oder im Fehlerfall –1.

rastPort: Adresse der RastPort-Struktur.
x: Neue X-Koordinate.
y: Neue Y-Koordinate.

AskFont(rastPort,textAttr)

rastPort, textAttr: iexp

Diese Funktion füllt die TextAttr-Struktur mit den Textattribut-Parametern des Rastport-Zeichensatzes.

rastPort: Adresse der RastPort-Struktur.
textAttr: Adresse der TextAttr-Struktur.

AskSoftStyle(rastPort)

rastPort: iexp

Ermittelt den Stil des Rastport-Zeichensatzes. Rückgabe: normal=00000000 (0), unterstrichen =00000001 (1) , fett =00000010 (2), kursiv =00000100 (4), gedehnt =00001000 (8)

rastPort: Adresse der RastPort-Struktur

BltBitMap(srcBitMap,srcX,srcY,destBitMap,destX,destY,sizeX, sizeY,minTerm,mask,tempA)

srcBitMap, srcX, srcY, destBitMap, destX: i exp
destY, sizeX, sizeY, minTerm, mask, tempA: iexp

Der Blitter kopiert Quell-BitMap-Inhalte in Ziel-BitMap-Inhalte. Rückgabe: Anzahl der Bitplanes die kopiert wurden.

srcBitMap:	Adresse der Quell-BitMap-Struktur.
srcX:	X-Koordinate der linken oberen Ecke in Pixel.
srcY:	Y-Koordinate der linken oberen Ecke in Pixel.
destBitMap:	Adresse der Ziel-BitMap-Struktur.
destX:	X-Koordinate der linken oberen Ecke in Pixel.
destY:	Y-Koordinate der linken oberen Ecke in Pixel.
sizeX:	Breite in Pixel.
sizeY:	Höhe in Pixel.
minTerm:	Logische Blitter-Verknüpfung.
mask:	Blitter-Maske. Zu verknüpfende Bitplanes.
tempA:	Adresse eines Chip-Speicher-Puffers (bei überlappenden Rechtecken).

BltBitMapRastPort(sourceBitMap,sourceX,sourceY,destRastPort, destX,destY,sizeX,sizeY,minTerm)

sourceBitMap, sourceX, sourceY, destRastPort: iexp
destX, destY, sizeX, sizeY, minTerm: iexp

Der Blitter kopiert Quell-BitMap-Inhalte in die BitMap einer RastPort-Struktur.

sourceBitMap:	Adresse der Quell-BitMap-Struktur.
sourceX:	X-Koordinate der linken oberen Ecke in Pixel.
sourceY:	Y-Koordinate der linken oberen Ecke in Pixel.
destRastPort:	Adresse der RastPort-Struktur, die die Adresse der Ziel-BitMap beinhaltet.
destX:	X-Koordinate der linken oberen Ecke in Pixel.
destY:	Y-Koordinate der linken oberen Ecke in Pixel.
sizeX:	Breite des Kopierbereichs.
sizeY:	Höhe des Kopierbereichs.

minTerm: Logische Blitter-Verknüpfung.

BltClear(memBlock,bytecount,flags)

memBlock, bytecount, flags: iexp

Der Blitter wird angewiesen einen Chip-Speicherbereich mit 0 zu füllen.

memBlock: Adresse des Chip-Speicherbereichs.
flags: Ist Bit 1 gesetzt, soll der „bytesperrow-Modus" genutzt werden Ist Bit 0 gesetzt, wartet die Funktion bis der Blitter fertig ist.
bytecount: Anzahl der Bytes. Ist Bit 1 der flags nicht gesetzt, wird an bytecount die Speichergröße in Bytes zugewiesen. Andernfalls müssen die unteren 16 Bit von Bytecount die Anzahl der Bytes pro Zeile und die oberen 16 Bit die Anzahl der Zeilen, enthalten. Dies kann eingesetzt werden, wenn der Inhalt von Bitplanes gelöscht werden soll.

BltMaskBitMapRastPort(sourceBitMap,sourceX,sourceY, destRastPort,destX,destY,sizeX,sizeY,minterm,bltMask)

sourceBitMap, sourceX, sourceY, destRastPort: iexp
destX, destY, sizeX, sizeY, minterm, bltMask: iexp

Der Blitter kopiert den Quell-BitMap-Inhalt in die BitMap eines Rastports und verknüft diese mit der angegebenen Maske.

sourceBitMap: Adresse der BitMap-Struktur.
sourceX: X-Koordinate der linken oberen Ecke in Pixel.
sourceY: Y-Koordinate der linken oberen Ecke in Pixel.
destRastPort: Adresse der RastPort-Struktur welche die Adresse der Ziel-BitMap Struktur beinhaltet.
destX: X-Koordinate der linken oberen Ecke in Pixel.
destY: Y-Koordinate der linken oberen Ecke in Pixel.
sizeX: Breite des Kopierbereichs.
sizeY: Höhe des Kopierbereichs.
minterm: Logische Blitter-Verknüpfung.
bltMask: Blitter-Maske. Zu verknüpfende Bitplanes.

BltPattern(rastPort,maskBitMap,x1,y1,maxx,maxy,bytecount)

rastPort, maskBitMap: iexp
x1, y1, maxx, maxy, bytecount: ivar

Verknüpft eine Rastport-Bitmap mit einer Maskier-Bitmap.

rastPort:	Adresse der RastPort-Struktur.
maskBitMap:	Adresse der Bitmapmaske.
x1:	X-Koordinate der linken oberen Ecke in Pixel.
y1:	Y-Koordinate der linken oberen Ecke in Pixel.
maxx:	Breite des Kopierbereichs.
maxy:	Höhe des Kopierbereichs.
bytecount:	Anzahl der Bytes pro Zeile.

BltTemplate(source,srcX,srcModulo,destRastPort,destX,destY, sizeX,sizeY)

source, srcX, srcModulo, destRastPort, destX, destY, sizeX, sizeY:iexp

Kopiert einen Speicherbereich in eine Bitmap.

source:	Adresse des Quell-Bitarrays.
srcX:	X-Koordinate der Quelle.
srcModulo:	Distanz zur nächsten Zeile im Array.
destRastPort:	Adresse der Ziel-Rast-Port Struktur.
destX:	X-Koordinate innerhalb der Bitmap.
destY:	Y-Koordinate innerhalb der Bitmap.
sizeX:	Breite des Kopierbereichs.
sizeY:	Höhe des Kopierbereichs.

ChangeSprite(viewPort,simpleSprite,spriteImage)

viewPort, simpleSprite, spriteImage: iexp

Ändert die Spritedaten.

viewPort:	Adresse der ViewPort-Struktur.
simpleSprite:	Adresse der SimpleSprite-Struktur.
spriteImage:	Adresse der SpriteImage-Struktur.

ClearEOL(rastport)

rastport: iexp

Löscht ein Textrechteck.

rastPort: Adresse der RastPort-Struktur.

ClearRectRegion(region,rectangle)

region, rectangle: iexp

Alle Fläschen des Bereichs (region), die innerhalb des Clip-Rechtecks (rectangle) liegen, werden gelöscht. Rückgabe: TRUE oder FALSE.

region: Adresse der Region-Struktur.
rectangle: Adresse der Rectangle-Struktur.

ClearRegion(region)

region: iexp

Diese Funktion löscht alle Clipping-Rechtecke eines Grafikbereichs (region).

region: Adresse der Region-Struktur.

ClipBlit(srcRastPort,srcX,srcY,destRastPort,destX,destY,sizeX, sizeY,minTerm)

srcRastPort, destRastPort: iexp
srcX, srcY, destX, destY, sizeX, sizeY, minTerm: ivar

Mit dem Blitter wird Rastport-Bitmap-Rechteck in eine andere Rastport-Bitmap kopiert.

srcRastPort: Adresse der Quell-RastPort-Struktur.
srcX: X-Koordinate der linken oberen Ecke.
srcY: Y-Koordinate der linken oberen Ecke.
destRastPort: Adresse der Ziel-RastPort-Struktur.
destX: X-Koordinate der linken oberen Ecke.
destY: Y-Koordinate der linken oberen Ecke.

sizeX: Breite des Kopierbereichs.
sizeY: Höhe des Kopierbereichs.
minTerm: Logische Blitter-Verknüpfung.

CopySBitMap(layer)

layer: iexp

Kopiert den Inhalt einer Superbitmap, je nach Größe des Layers, in eine Layerbitmap. Dazu muß der Layer als Superbitmaplayer definiert sein.

layer: Adresse der Layer-Struktur.

CWait(clist,xbeam,ybeam)

clist: iexp
xbeam, ybeam: ivar

CWait fügt eine neue Instruktion an die Copper-Intuition-Liste an und führt diese aus, wenn der Elektronenstrahl die Position xbeam, ybeam erreicht hat.

clist: Adresse der UCopList-Struktur.
xbeam, ybream: Koordinate des Elektronenstrahl.

DisownBlitter()

Diese Funktion gibt den Blitter für andere Tasks frei. Keine Rückgabe.

DisposeRegion(region)

region: iexp

Gibt den gesamten Speicher der Region-Struktur und der zugehörigen Rectangle-Strukturen frei.

region: Adresse der Region-Struktur.

DoCollision(rastport)

rastport: iexp

Testet Grafikelemente auf Kollision.

rastport: Adresse der RastPort Struktur.

Draw(rastport,x,y)

rastport: iexp
x, y: ivar

Zeichnet eine Linie ab der aktuellen Grafikcursorposition zu der in x und y bestimmten Position.

rastport: Adresse der Rastport-Struktur.
x, y: Zeichenkoordinate

DrawCircle(rastport,x,y,radius)

rastport: iexp
x, y, radius: ivar

Zeichnet einen Kreis in den für den angegebenen Rastport voreingestellten Zeichenmodi und Farben.

rastport: Adresse des Ausgabe-Rastports.
x, y: Kreismittelpunkt.
radius: Radius des Kreises.

DrawEllipse(rastport,x,y,x_radius,y_radius)

rastport: iexp
x, y, x_radius, y_radius: ivar

Zeichnet eine Ellipse.

rastport: Adresse des Ausgabe-Rastports.
x, y: Mittelpunkt-Koordinate.
x_radius: Horizontaler Radius.
y_radius: Vertikaler Radius.

DrawGList(rastport,viewport)

rastport, viewport: iexp

Die Sprites und Bobs der aktuellen Grafik-Elementen-Liste sollen gezeichnet werden.

rastport: Adresse des Ausgabe-Rastports.
viewport: Adresse des Viewport-Struktur, welche die Copper-Befehle zur Darstellung der Sprites beinhaltet.

Flood(rastport,mode,x,y)

rastport, mode: iexp
x, y: ivar

Flood füllt einen Rastport-Bereich mit dem aktuellen Füllmuster in der aktuellen Vorder- und Hintergrundfarbe. Keine Rückgabe.

rastport: Adresse der RastPort-Struktur.
mode: Bei mode=1, wird nur der Bereich gefüllt, der die gleiche Farbe hat, wie der Punkt an der Koordinate x,y.
Bei mode=0, wird der Füllbereich durch die mit der Funktion SetOPen gesetzten Farbe begrenzt.
x, y: Startkoordinate bei Füllbeginn.

FreeColorMap(colormap)

colormap: iexp

Gibt den Speicher, der durch eine ColorMap Struktur reserviert wurde wieder frei. Keine Rückgabe.

colormap: Adresse der ColorMap-Struktur

FreeCopList(coplist)

oplist: iexp

Diese Funktion gibt den durch die CopList-Struktur reservierten Speicher wieder frei. Keine Rückgabe.

coplist: Adresse der CopList-Struktur.

FreeCprList(cprlist)

cprlist: iexp

Diese Funktion gibt den durch die CprList-Struktur reservierten Speicher wieder frei. Keine Rückgabe.

cprlist: Adresse der CprList-Struktur.

FreeGBuffers(animob,rastport,buf)

animob, rastport, buf: iexp

Die Funktion löscht den Buffer der Animations-Objekte, der mit der Funktion GetGBuffers reserviert wurde. Keine Rückgabe.

animob: Adresse einer AnimOb-Struktur.
rastport: Adresse des Ausgabe-Rastports.
buf: Adresse des Buffers.

FreeSprite(spritenr)

spritenr: iexp

Löscht einen Hardware-Sprite. Keine Rückgabe.

spritenr: Spritenummer (0–7).

FreeVPortCopLists(viewport)

vieport: iexp

Diese Funktion löscht den Speicherbereich der durch alle Copper-Listen in dem angegebenen Viewport belegt wird.

GetColorMap(count)

count: ivar

GetColorMap reserviert Speicher und initialisiert eine ColorMap-Struktur. Rückgabe: Adresse der ColorMap-Struktur.

count: Anzahl der Farbregister

GetGBuffers(animob,rastport,buffer)

animob, rastport, buffer: iexp

Reserviert alle Zwischenspeicher die von Animations Objekten benötigt werden. Rückgabe: TRUE oder FALSE.

animob: Adresse einer AnimOb-Struktur.
rastport: Adresse des Ausgabe-Rastports.
buffer: Bei Angabe von TRUE wird doppelt zwischengespeichert.

GetRGB4(cmap,regnr)

cmap: iexp
regnr: ivar

Liest den Farbwert eines Farbregisters. Rückgabe: Farbwert des angegebenen Registers.

cmap: Adresse der ColorMap.
regnr: Registernummer (0–63).

GetScreenData(buffer,größe,typ,screen)

buffer, type, screen: iexp
größe: ivar

Diese Funktion kopiert einen Teil oder den ganzen Inhalt einer Screen Struktur. Die Funktion kann dazu verwendet werden, die Größe und Art von Screen zu erfragen. Dabei handelt es sich meist um den Workbench-Screen. Rückgabe: TRUE oder FALSE

buffer:	Adresse des Speicherbereichs in den die Screen Daten geschrieben werden sollen.
größe:	Die Größe des Speicherbereichs in Bytes.
type:	WBENCHSCREEN (1) oder CUSTOMSCREEN (15).
screen:	Adresse eines CUSTOMSCREENS. Wird bei einer Typangabe von 1 ignoriert.

GetSprite(spriteadr,spritenr)

spriteadr: iexp
spritenr: ivar

Reserviert einen Hardware-Sprite. Der Sprite wird dabei nicht auf dem Bildschirm angezeigt. Rückgabe: Nummer des Sprites, der gesetzt wurde oder –1, wenn keine weiteren Sprites gesetzt werden können.

spriteadr:	Adresse einer SimpleSprite-Struktur.
spritenr:	Nummer des Sprites, der gesetzt werden soll (0–7).

InitArea(info,buffer,maxkoord)

info, buffer: iexp
maxkoord: ivar

Initialisiert eine Area-Liste. Mit Hilfe von Area-Listen können mehrer Grafikelemente direkt auf dem Bildschirm angezeigt werden. Keine Rückgabe.

buffer:	Adresse des Speicherbereichs für die Area-Listen.
maxkoord:	Maximale Anzahl der Koordinaten in der Liste.

InitBitMap(bitmap,tiefe,weite,höhe)

bitmap: iexp
tiefe, weite, höhe: ivar

Initialisiert eine Bitmap. Die einzelnen Bitplanes der Bitmap müssen mit der Funktion AllocRaster() nach dem Aufruf von InitBitMap zusätzlich noch reserviert werden. Keine Rückgabe.

bitmap:	Adresse der BitMap-Struktur.
tiefe:	Bitplanetiefe der BitMap.
weite, höhe:	Größe der Bitmap.

InitGels(sprite1,sprite2,info)

sprite1, sprite2, info: iexp

Initialisiert eine Grafikelement-Liste für VSprites. Diese Funktion muß vor der Benutzung von VSprites einmal aufgerufen werden. Keine Rückgabe.

sprite1:	Adresse einer leeren VSprite-Struktur, die den Anfang der Liste kennzeichnet.
sprite2:	Adresse einer leeren VSprite-Struktur, die das Ende der Liste kennzeichnet.
info:	Adresse einer GelsInfo-Struktur, die initialisiert werden soll.

InitGMasks(animob)

animob: iexp

InitGMask initialisiert alle Masken einer AnimOb-Struktur. Keine Rückgabe.

animob: Adresse einer AnimOb-Struktur.

InitMasks(sprite)

sprite: iexp

Die Kollisionsgröße des Sprites wird ermittelt und in die VSprite Struktur eingetragen. Keine Rückgabe.

sprite: Adresse einer VSprite-Struktur.

InitRastPort(rastport)

rastport: iexp

Diese Funktion initialisiert eine RastPort-Struktur. Alle Variablen der RastPort-Struktur werden auf 0 gesetzt. Der Wert für FgPen, AOlPen und LinePtn ist danach –1. Der Zeichenmodus JAM2 und der Zeichenfont je nach Auflösung Topaz 80 oder Topaz 60. Keine Rückgabe.

rastport: Adresse der RastPort-Struktur.

InitTmpRas(tmpras,buffer,größe)

tmpras, buffer: iexp
größe: ivar

Diese Funktion bereitet einen Chip-Speicherbereich auf die Verwendung durch Area-Funktionen vor. Dieser Speicherbereich wird von allen Area-Funktionen benötigt und muß unbedingt angefordert werden. Keine Rückgabe.

tmpras: Adresse einer TmpRas-Struktur.
buffer: Adresse des Chip-Speichers, der von den Area-Funktionen genutzt werden soll.
größe: Größe des Speicherbereichs in Byte.

InitView(view)

view: iexp

InitView füllt eine View-Struktur mit fest voreingestellten Werten. Keine Rückgabe.

view: Adresse einer View-Struktur.

InitVPort(viewport)

viewport: iexp

Diese Funktion setzt alle Variablen einer View-Struktur auf 0. Keine Rückgabe.

viewport: Adresse der ViewPort-Struktur, die initialisiert werden soll.

InstallClipRegion(layer,region)

layer, region: iexp

InstallClipRegion definiert ein Clip-Rechteck innerhalb eines Layers. Alle Grafikausgaben auf diesem Layer werden dann nur noch innerhalb dieses Clip-Rechteckes sichtbar. Rückgabe: Adresse der letzten Region-Struktur oder 0.

layer: Adresse der Layer-Struktur.
region: Adresse der Region-Struktur, die die Koordinaten des Clip-Rechteckes enthält.

LoadRGB4(viewport,colors,count)

viewport, colors: iexp
count: ivar

Mit dieser Fuktion kann eine Anzahl Farben an einen Viewport übergeben werden. Keine Rückgabe.

viewport: Adresse der ViewPort-Struktur, an die die Farbpalette zugewiesen werden soll.

colors: Adresse, ab der die Farbpalette im Speicher steht.
count: Anzahl der Farbregister, die zu belegen sind.

LoadView(view)

view: iexp

Stellt die in der Copper-Liste enhaltenen Informationen auf dem in „view" angegeben Bildschirm dar. Keine Rückgabe.

view: Adresse der View-Struktur.

LockIBase(nummer)

nummer: iexp

Wird diese Funktion ausgeführt, so kann weder Intuition noch irgendein anderes Programm Veränderungen an der IntuitionBase-Struktur vornehmen. Rückgabe: Vier Byte Integer-Wert, der zum späteren Freigeben der Struktur mit der Funktion UnLockIBase() gebraucht wird.

nummer: Reserviert für spätere Versionen. Muß 0 enthalten.

MakeVPort(view,viewport)

view, viewport: iexp

Mit dieser Funktion können Dual Playfields erzeugt werden. Dazu muß der ersten RasInfo-Struktur der ViewPort-Struktur, die Adresse einer zweiten RasInfo-Struktur angehängt werden. Keine Rückgabe.

view: Adresse der View-Struktur.
viewport: Adresse der ViewPort-Struktur.

MenuNum(code)

code: iexp

Ermittelt die Nummer des angewählten Menüs. Rückgabe: Nummer des Menüs.

code: Der Wert der Code-Variablen einer IntuiMessage-Struktur.
Oder der Wert von MENU(2).

Move(rastport,x,y)

rastport: iexp
x, y: ivar

Stellt den Grafikcursor an eine neue Position innerhalb des Ausgabe-Rastports. Keine Rückgabe.

rastport: Adresse der Rastport-Struktur.
x, y: Neue Koordinate für den Grafikcursor.

MoveSprite(viewport,spriteadr,x,y)

viewport, spriteadr: iexp
x, y: ivar

Bewegt einen Hardware-Sprite zu der in x und y angegeben Position. Keine Rückgabe.

viewport: Adresse der ViewPort-Struktur.
spriteadr: Adresse der SimpleSprite-Struktur.
x, y: Neue Sprite-Koordinate.

MrgCop(view)

view: iexp

Füllt die Copper-Instruktionsliste mit neuen Anweisungen. Keine Rückgabe.

view: Adresse der View-Struktur.

NewRegion()

NewRegion initialisiert ein neues Clip-Rechteck mit der Größe 0. Rückgabe: Adresse der neuen Region-Struktur.

OrRectRegion(region,cliprect)

region, cliprect: iexp

Diese Funktion führt eine Oder-Verknüpfung zwischen einem Grafikbereich und einem Clipbereich durch. Das Ergebnis dieser Verknüpfung wird im Grafikbereich angezeigt. Keine Rückgabe.

region: Adresse der Region-Struktur.
cliprect: Adresse der Rectangle-Struktur.

OrRegionRegion(region1,region2)

region1, region2: iexp

Diese Funktion führt eine Oder-Verknüpfung zwischen zwei Grafikbereichen durch. Das Ergebnis dieser Verknüpfung wird im zweiten Grafikbereich angezeigt. Keine Rückgabe

region1: Adresse der ersten Region-Struktur.
region2: Adresse der zweiten Region-Struktur.

OwnBlitter()

Diese Funktion verbietet anderen Programmen den Zugriff auf dem Blitter-Chip.

PolyDraw(rastport,count,kooradr)

rastport, koordadr: iexp
count: ivar

Zeichnet ein Polygon mit „count“ Ecken. Keine Rückgabe.

count: Anzahl der Koordinatenpaare.

koordadr: Adresse des Speicherbereichs, der die Koordinaten enthält.

QBlit(blitnode)

blitnode: iexp

QBlit fügt eine BlitNode-Struktur an das Ende der aktuellen Blitter-Liste und führt die Blitter-Routine aus, die in der BlitNode-Struktur angegeben ist. Keine Rückgabe.

blitnode: Adresse einer BlitNode-Struktur.

QBSBlit(blitnode)

blitnode: iexp

QBSBlit synchronisiert die auszuführende Blitter-Rountine zusätzlich noch mit dem Strahrücklauf. Keine Rückgabe.

blitnode: Adresse einer BlitNode-Struktur.

RasSize(w,h)

w, h: ivar

Diese Funktion berechnet die Bytes die für eine Raster-Bitplane gebraucht werden. Rückgabe: Größe in Byte.

w, h: Die Weite und Bitplane

ReadPixel(rastport,x,y)

rastport: iexp
x, y: ivar

Ermittelt die Farbregisternummer der Farbe des an der angegebenen Koordinate stehenden Pixels. Rückgabe: Farbnummer oder –1.

rastport: Rastport-Struktur des Rastports, in dem die Pixelfarbe ausgelesen werden soll.
x, y: Koordinate des Pixel.

RectFill(rastport,x1,y1,x2,y2)

rastport: iexp
x1, y1, x2, y2: ivar

Die Funktion RectFill zeichnet ein ausgefülltes Rechteck im aktuellen Füllmuster, Zeichenmodi und Zeichenfarbe. Keine Rückgabe.

rastport:	Adresse der Ausgabe-RastPort-Struktur.
x1, y1:	Linke obere Ecke des Rechtecks.
x2, y2:	Rechte untere Ecke des Rechtecks.

RemBob(bob,rastport,viewport)

bob, rastport, viewport: iexp

RemBob löscht eine Bob-Struktur aus der Grafik-Elemente-Liste. Keine Rückgabe.

bob:	Adresse einer Bob-Struktur.
rastport:	Adresse einer Rastport-Struktur.
viewport:	Adresse einer ViewPort-Struktur.

RemVSprite(sprite)

sprite: iexp

Entfernt eine VSprite-Struktur aus der Grafik-Elemente-Liste. Keine Rückgabe.

sprite:	Adresse der VSprite-Struktur.

ReportMouse(window,wahr)

window, wahr: iexp

Mit dieser Funktion kann die Überwachung der Mauskoordinaten in einem Fenster ein- oder ausgeschaltet werden. Keine Rückgabe.

window:	Adresse der Window Struktur.
wahr:	Wahrheitswert zum Ein- oder Ausschalten.

ScrollRaster(rastport,deltax,deltay,x,y,w,h)

rastport: iexp
deltax, deltay, x, y, w, h: ivar

Scrollt einen Ausschnitt eines Rastports. Keine Rückgabe.

rastport:	Adresse der RastPort Struktur.
dx, dy:	Anzahl der Pixel,die gescrollt werden sollen.
x, y:	Obere linke Ecke des Scroll-Bereichs.
w, h:	Untere rechte Ecke des Scroll-Bereichs.

ScrollVPort(viewport,deltax,deltay)

viewport: iexp
deltax, deltay: ivar

Mit dieser Funktion kann der ganze Inhalt eines Screens gescrollt werden. Keine Rückgabe.

viewport:	Adresse der ViewPort-Struktur des Screens.
deltax, deltay:	Neue Kordinate, relativ zur alten.

SetAfPt(rastport,data,anz)

rastport, data: iexp
anz: ivar

Setzt ein Füllmuster für Area-Funktionen und die RectFill() Funktion. Keine Rückgabe.

rastport:	Adresse einer RastPort-Struktur.

data: Adresse des Speichers, in der die Füllmuster-Daten stehen.
anz: 2^anz, oder Anzahl der Worte in diesem Datenbereich.

SetAPen(rastport,nr)

rastport: iexp
nr: ivar

Setzt die Vordergrundzeichenfarbe. Keine Rückgabe.

rastport: Adresse der RastPort-Stuktur
nr: Farbregisternummer.

SetBPen(rastport,nr)

rastport: iexp
nr: ivar

Setzt die Hintergrundzeichenfarbe. Keine Rückgabe.

rastport: Adresse der RastPort-Stuktur
nr: Farbregisternummer.

SetCollision(nummer,routine,gelsinfo)

nummer: ivar
routine, gelsinfo: iexp

Mit dieser Funktion wird die Adresse der Funktion übergeben die bei der Kollision von Sprites angesprungen werden soll. Keine Rückgabe.

nummer: Die Nummer der Funktion (0–15).
routine: Adresse der Funktion.
gelsinfo: Adresse einer GelsInfo-Struktur.

SetDrMd(rastport,mode)

rastport: iexp
mode: ivar

Setzt den Zeichenmodus. Keine Rückgabe.

rastport: Adresse der Rastports.
mode: Zeichenmodus (wie bei GRAPHMODE).

SetDrPt(rastport,data)

rastport, data: iexp

Setzt ein spezielles Linien-Muster. Keine Rückgabe.

rastport: Adresse der RastPort-Struktur.
data: 16-Bit-Wert, der das Aussehen der Linie enthält.

SetFont(rastport,textfont)

Diese Funktion ordnet dem angegebenen Rastport einen bestimmten Zeichensatz zu. Rückgabe: 0, wenn kein Fehler aufgetreten ist.

rastport: Adresse der RastPort-Struktur.
textfont: Adresse der TextFont-Struktur.

SetOPen(rastport,register)

rastport: iexp
register: ivar

Diese Funktion setzt die Umrahmungs-Farbe für Area-Funktionen. Keine Rückgabe.

rastport: Adresse der RastPort-Struktur.
register: Farbregisternummer.

SetRast(rastport,register)

rastport: iexp
register: ivar

Füllt den gesamten Rastport mit der angegebenen Farbe. Keine Rückgabe.

rastport:	Adresse der RastPort-Struktur.
regsiter:	Farbregisternummer.

SetRGB4(viewport,register,rot,grün,blau)

viewport: iexp
register, rot, grün, blau: ivar

Setzt ein Farbregister eines Viewports auf den spezifizierten Farbwert. Keine Rückgabe.

viewport:	Adresse der ViewPort-Struktur.
register:	Farbregisternummer.
rot, grün, blau:	Die jeweiligen Farbwerte (0–15).

SetRGB4CM(colormap,register,rot,grün,blau)

colormap: iexp
register, rot, grün, blau: ivar

Setzt ein Farbregister einer ColorMap auf den spezifizierten Farbwert. Keine Rückgabe.

colormap:	Adresse der ColorMap-Struktur.
register:	Farbregisternummer.
rot, grün, blau:	Die jeweiligen Farbwerte (0–15).

SetSoftStyle(rastport,neu,maske)

rastport, neu, maske: iexp

Setzt einen neuen Darstellungsmodus für den, mit der Funktion SetFont() gesetzten, Zeichensatz. Rückgabe: Der gleiche Wert wie „maske“.

rastport: Adresse der RastPort-Struktur.
neu: Der neue Darstellungsmodus. (siehe AskSoftStyle)
maske: Gibt an, welche Bits verändert werden dürfen.

SortGList(rastport)

rastport: iexp

Sortiert die VSprite-Liste nach Y-Koordinaten. Keine Rückgabe.

rastport: Adresse der RastPort-Struktur.

Text(rastport,text,länge)

rastport: iexp
text: sexp
länge: ivar

Schreibt den angegebenen Text in den Rastport, an die aktuelle Position des Grafikcursors. Rückgabe: 0, wenn kein Fehler auftrat.

rastport: Adresse der RastPort-Struktur.
text: Auszugebender Textstring.
länge: Länge des Textstrings.

TextLength(rastport,text,länge)

Diese Funktion berechnet die Länge eines Textstrings in Pixel. Die Länge des Strings bezieht sich dabei auf die Angaben in der TextFont-Struktur der RastPort-Struktur. Rückgabe: Die Länge des Strings in Pixel.

rastport: Adresse der RastPort-Struktur.
text: Auszugebender Textstring.
länge: Länge des Textstrings.

Translate(text,länge,phone,outlänge)

text: sexp
länge, phone, outlänge: iexp

Übersetzt den angegebenen Text in einen phonetischen Text für die Sprachausgabe.

text:	Zu übersetzender Textstring.
länge:	Länge des Textstrings.
phone:	Adresse der Speicherbereichs, in den die Übersetzung geschrieben werden soll.
outlänge:	Länge des Speicherbereichs.

UCopperListInit(clist,instnr)

clist: iexp
instnr: ivar

Diese Copper-Funktion initialisiert eine UCopList-Struktur. Diese Struktur wird benutzt, um dem Copper Anweisungslisten zu übermitteln.

clist:	Adresse der UCopList-Struktur.
instnr:	Die maximale Anzahl der enthaltenen Instruktionen.

VBeamPos()

Ermittelt die momentane Y-Position des Elektronenstrahls in der Bildröhre. Rückgabe: Y-Position.

ViewAddress()

Diese Funktion ermittelt die Adresse einer View-Struktur. Rückgabe: Adresse der View-Struktur.

ViewPortAddress(window)

window: iexp

Ermittelt die Adresse der Fenster-ViewPort-Struktur. Rückgabe: Adresse der ViewPort-Struktur.

WaitBlit()

Diese Funktion hält alle Tasks solange an, bis der Blitter seine Arbeit beendet hat.

WaitBOVP(viewport)

viewport: iexp

Wartet bis der Elektronenstrahl den Anfang des angegebenen Screens erreicht hat.

viewport: Adresse der ViewPort-Struktur.

WritePixel(rastport,x,y)

rastport: iexp
x, y: ivar

Diese Funktion setzt einen einzelnen Punkt, in der mit SetAPen gesetzten Farbe an die angegebene Position.

rastport: Adresse der RastPort-Struktur.
x, y: Zeichenkoordinate.

XorRectRegion(region,rect)

region, rect:iexp

Diese Funktion macht eine Exklusiv-Oder-Verknüpfung zwischen einem Clip-Rechteck und einem Grafikbereich. Das Ergebnis wird im Grafikbereich angezeigt.

region: Adresse der Region-Struktur.

rect: Adresse der ClipRect-Struktur.

XorRegionRegion(region1,region2)

region1, region2: iexp

Diese Funktion macht eine Exclusiv-Oder-Verknüpfung zwischen zwei Grafikbereichen. Das Ergebnis wird im zweiten Grafikbereich angezeigt.

region1: Adresse der ersten Region-Struktur.
region2: Adresse der zweiten Region-Struktur.

BumpRevision(newBuf,oldName)

newBuf, oldName: sexp

Mit dieser Funktion können kopierte Dateien umbenannt werden. Rückgabe: Größe des neuen Namen.

newBuf: Textadresse des neuen Namens (Maximal 31 Byte; 0-Byte am Ende).
oldName: Textadresse des des alten Namens (Maximal 31 Byte; 0-Byte am Ende).

CloseWorkBench()

Schließt den Workbench-Screen. Rückgabe: TRUE oder FALSE.

FindToolTypes(tools,typename)

tools, typename: iexp

Diese Funktion durchsucht einen ToolType-Bereich nach dem im „typename" angegebenen Namen. Rückgabe: Adresse des ersten gefunden Toolnames.

tools: Adresse des ToolType-Bereichs.
typename: Typname.

FreeDiskObject(object)

object: iexp

Löscht den Speicher der von einer Info-Datei belegt wird. Keine Rückgabe.

object: Adresse einer DiskObject-Struktur.

FreeWBObject(wbobject)

wbobject: iexp

Alle Eintragungen einer WBObject-Struktur und die Struktur selbst wird gelöscht. Keine Rückgabe.

wbobject: Adresse der WBObject-Struktur.

GetDiskObject(name)

name: sexp

Lädt eine Diskobject-Struktur einer Datei von Diskette. Rückgabe: Adresse der geladenen Diskobject-Struktur.

name: Name der Datei, dessen Object-Struktur geladen werden soll.

GetIcon(name,diskobject,freelist)

name: sexp
diskobject, freelist: iexp

Diese Funktion ist gleich der Funktion GetDiskObject(). Hierbei muß der Programmierer jedoch selbst eine FreeList-Struktur suchen. Rückgabe: TRUE oder FALSE

name: Name der Datei deren DiskObject-Struktur geladen werden soll.
diskobject: Adresse einer initialisierten DiskObject-Struktur.
freelist: Adresse der FreeList-Struktur aus der Speicher entnommen werden soll.

GetWBObject(name)

name: sexp

Liest Informationen eines Workbench-Objektes in eine WBObject-Struktur. Rückgabe: Adresse der WBObject-Struktur

name: Name der Datei von der die Informationen gelesen werden soll.

MatchToolValue(stradr1,stradr2)

stradr1, stradr2: iexp

MatchToolValue durchsucht eine Bereich von ToolType-Namen nach dem angegebenen Namen. Rückgabe: TRUE oder FALSE

stradr1: Adresse des ToolType-Namensbereichs.
stradr2: Adresse des Namens, nach dem gesucht werden soll.

OpenWorkBench()

Diese Funktion versucht den Workbench-Screen zu öffnen. Rückgabe: TRUE oder FALSE.

PutDiskObject(name,diskobject)

name: sexp
diskobject: iexp

Speichert die DiskObject-Struktur in der Info-Datei einer Datei ab. Rückgabe: TRUE oder FALSE

name: Name der Datei.
diskobject: Adresse der DiskObject-Struktur.

PutIcon(name,diskobject)

name: sexp
diskobject: iexp

Speichert die DiskObject-Struktur in der Info-Datei einer Datei ab. Rückgabe: TRUE oder FALSE

name: Name der Datei.
diskobject: Adresse der DiskObject-Struktur.

PutWBObject(name,wbobject)

name: sexp
wbobject: iexp

Speichert die WBObject-Struktur in der Info-Datei einer Datei ab. Rückgabe: TRUE oder FALSE

name:	Name der Datei.
wbobject:	Adresse der WBObject-Struktur.

WBenchToBack()

Der Workbench-Screen wird in den Hintergrund verlagert, so daß er von anderen Screens verdeckt wird.

WBenchToFront()

Zeigt den Workbench-Screen im Vordergrund an.

ActivateGadget(gadget,window,requester)

gadget, window, requester: iexp

Diese Funktion aktiviert ein String-Gadget in einem Fenster oder Requester. Rückgabe: Bei Erfolg TRUE, andernfalls FALSE.

gadget: Adresse der String-Gadget-Struktur.
window: Adresse der Window-Struktur, in welchem das Gadget steht.
requester: Adresse der Requester-Struktur oder 0.

ActivateWindow(window)

window: iexp

Mit dieser Funktion wird ein Window aktiviert.

window: Adresse der Window-Struktur.

AddGadget(pointer,gadget,position)

pointer, gadget, position: iexp

Mit dieser Funktion kann ein Gadget in die Gadget-Liste eines Screen oder Fensters eingefügt werden. Rückgabe: Position des Gadgets in der Liste.

pointer: Adresse des Screen- oder Window-Struktur.
gadget: Adresse der Gadget-Struktur.
position: Index des Gadgets in der Liste. 0 ist das erste Gadget.

AddGList(window,gadget,position,number_gadgets,requester)

window, gadget, position, number_gadgets, requester: iexp

Mit dieser Funktion können mehrere Gadgets in die Gadget-Liste eines Fensters oder Requesters eingefügt werden. Rückgabe: Index der Gadget-Liste.

window: Adresse der Fenster-Struktur.
gadget: Adresse des ersten Gadgets der Gadget-Liste.
position: Gadget-Index an der die Gagdet-Liste eingefügt werden soll.

number_gadgets: Anzahl der einzufügenden Gadgets.
requester: Adresse der Requester-Struktur.

AllocRemember(rememberKey,size,art)

rememberKey, size, art:iexp

Diese Funktion reserviert Speicher und verknüpft ihn in einer Liste. Die einzelnen Speicherbereiche in der Liste können mit der Funktion FreeRemember(), komplett freigegeben werden. Rückgabe: Adresse auf den Speicherbereich oder 0.

rememberKey: Adresse der Remember-Struktur. Beim ersten Aufruf 0.
size: Speichergröße in Bytes.
art: Speicherattribute.

AlohaWorkbench(msgport)

msgport: iexp

AlohaWorkbench schaltet die Workbench-Überwachung für Intuition ein. Sobald die Workbench angeklickt wird, erhält man im angegebenen MessagePort die Nachricht (Class) WBENCHMESSAGE. Keine Rückgabe.

msgport: Adresse der MessagePort-Struktur, die die Nachricht erhalten soll.

AutoRequest(wind,bodText,posText,negText,posFlage,negFlags, width,height)

wind, bodText, posText, negText, posFlage, negFlags, width, height:iexp

Diese Funktion öffnet einen System-Requester und wartet auf Benutzeraktivitäten. Rückgabe: TRUE oder FALSE.

wind: Adresse der Window-Struktur.
bodText: Adresse der IntuiText-Struktur.
posText: Adresse der zweiten IntuiText-Struktur (OK Gadget).
negText: Adresse der dritten IntuiText-Struktur (Abbruch Gadget).
posFlage: IDCMP Flags.

negFlags: IDCMP Flags.
width: Breite des Requesters in Pixel.
height: Höhe des Requesters in Pixel.

BeginRefresh(window)

window: iexp

Diese Funktion wird für (simple-refresh) Windows benötigt und meldet Intuition den Start eines Windowrefresh. Außerdem wird die Funktion BeginUpdate() aufgerufen. Keine Rückgabe.

window: Adresse der Window-Struktur.

BuildSysRequest(window,bodyText,positivText,negativText, IDCMPFlags,width,height)

window, bodyText, positivText, negativText, IDCMPFlags: iexp
width, height: ivar

Diese Funktion öffnet einen Requester in einem Window und wartet auf Benutzeraktivitäten. Wird keine Fensteradresse angegeben so erzeugt Intuition ein Fenster für den Requester. Rückgabe: Adresse der Window-Struktur.

window: Adresse der Window-Struktur oder 0.
bodyText: Adresse der IntuiText-Struktur.
positiveText: Adresse der zweiten IntuiText-Struktur.
negativeText: Adresse der dritten IntuiText-Struktur.
IDCMPFlags: IDCMPFlags für das Fenster.
width: Breite des Requester.
height: Höhe des Requester.

ClearDMRequest(window)

window: iexp

Diese Funktion löscht einen DoubleMenu-Requester des Windows. Rückgabe: TRUE oder FALSE.

window: Adresse der Window-Struktur.

ClearMenuStrip(window)

window: iexp

Löscht eine Menüleiste aus einem Window. Keine Rückgabe.

window: Adresse der Window-Struktur

ClearPointer(window)

window: iexp

Setzt das Aussehen des Mauszeigers, entsprechend dem in den System-Preferences enthaltenen Maus-Images. Keine Rückgabe.

window: Adresse der Window-Struktur.

ClearScreen(rastPort)

rastPort: iexp

Löscht den RastPort ab der aktuellen Grafikcursorposition. Keine Rückgabe.

rastPort: Adresse der RastPort-Struktur.

CloseScreen(screen)

screen: iexp

Schließt einen Screen und gibt den Speicher frei. Keine Rückgabe.

screen: Adresse der Screen-Struktur.

CloseWindow(window)

window: iexp

Schließt ein Fenster und gibt den Speicher frei. Keine Rückgabe.

window: Adresse der Window-Struktur.

CurrentTime(Seconds,Micros)

Seconds, Micros: iexp

Ermittelt die aktuelle Systemzeit in Sekunden und Millisekunden. Keine Rückgabe.

Seconds: Adresse der Variable für Sekunden.
Micros: Adresse der Variable für Millisekunden.

DisplayAlert(alertNumber,string,height)

alertNumber, height: iexp
string: sexp

Diese Funktion öffnet eine Alert-Box auf dem Screen. Rückgabe: TRUE oder FALSE.

alertNumber: Die Nummer der Alert-Mitteilung.
string: Nachrichten-String.
height: Anzahl der Zeilen.

DisplayBeep(screen)

screen: iexp

Läßt den Screen flackern und dient als sichtbare Warnung. Keine Rückgabe.

screen:Adresse der Screen-Struktur. Wird als Screen-Adresse 0 angegeben so flackern alle Screens.

DoubleClick(s_sek,s_mic,e_sek,e_mic)

s_sek, s_mic, e_sek, e_mic: ivar

DoubleClick prüft ob ein doppeltes Drücken der linken Maustaste stattgefunden hat. Die Entscheidung richtet sich nach den Einstellungen der System-Preferences. Die Start- und Endzeiten können mit der Funktion CurrentTime ermittelt werden. Rückgabe: War die Zeit des Doppelklicks zu lang wird FALSE zurückgeliefert, andernfalls TRUE.

s_sek: Startsekunde
s_mic: Startmikrosekunde
e_sek: Endsekunde
e_mic: Endmikrosekunde

DrawBorder(rastport,border,x_off,y_off)

rastport, border: iexp
x_off, y_off: ivar

Durch diese Funktion können ganze Border-Strukturen gezeichnet werden. Dabei wird die Verkettung der Strukturen untereinander beachtet. Keine Rückgabe.

rastport: Adresse des Ausgabe-Rastports.
border: Adresse der ersten Border-Stuktur.
x_off, y_off: Offsetkoordinate die zur Startkoordinate des Borders addiert werden soll.

DrawImage(rastport,image,x_off,y_off)

rastport, image: iexp
x_off, y_off: ivar

Zeichnet Images in einen Screen oder ein Fenster. Sind die Image-Strukturen untereinander verkettet, wird die komplette Image-Liste gezeichnet. Keine Rückgabe.

rastoprt: Adresse des Ausgabe-Rastports.
image: Adresse der ersten Image-Struktur.
x_off, y_off: Koordinate die zur Startkoordinate der Images hinzuaddiert werden soll.

EndRefresh(window,complete)

window, complete: iexp

Das Neuzeichnen eines Simple-Refresh-Fensters soll beendet werden. Dazu muß die Funktion BeginRefresh() vorausgegangen sein. Keine Rückgabe.

window: Adresse der Window-Struktur des gerade neu gezeichneten Fensters.
complete: TRUE oder FALSE. Je nachdem, ob die Window-Bitmap komplett oder nur teilweise neu gezeichnet wurde.

EndRequest(requester,window)

requester, window: iexp

Diese Funktion löschen den in „requester" angegebenen Requester aus dem Fenster. Keine Rückgabe.

requester: Adresse der Requester-Struktur.
window: Adresse der Window-Struktur.

FreeRemember(remember,flag)

remember, flag: iexp

Löscht entweder eine Remember-Struktur oder die Speicherbereiche, die in einer Struktur miteinander verkettet sind. Keine Rückgabe.

remember: Adresse der Remember-Struktur.
flag: Ist „flag" gleich 1, so werden die Speicherbereiche gelöscht. Andernfalls nur die Remember-Struktur.

FreeSysRequest(window)

window: iexp

Löscht alle Requester die mit der Funktion BuildSysRequest() erstellt wurden. Keine Rückgabe.

window: Adresse der Window-Struktur.

GetDefPrefs(buffer,byte)

buffer, byte:iexp

Liest die System-Preferences-Einstellungen von der Diskette. Keine Rückgabe.

buffer: Adresse des Buffer-Speichers, der vorher reserviert werden muß.
byte: Anzahl der Bytes, die kopiert werden sollen.

GetPrefs(buffer,byte)

Liest die System-Preferences-Einstellungen aus dem Speicher. Keine Rückgabe.

buffer: Adresse des Buffer-Speichers, der vorher reserviert werden muß.
byte: Anzahl der Bytes, die kopiert werden sollen.

InitRequester(requester)

requester: iexp

Initialisiert eine Requester-Struktur. Diese Funktion zeigt den Requester nicht auf dem Bildschirm an. Keine Rückgabe.

requester: Adresse der Requester-Struktur.

IntuiTextLength(intuitext)

intuitext: iexp

Diese Funktion berechnet die Länge eines IntuiText-Strings in Pixel. Die Länge des Strings bezieht sich dabei auf die Angaben in der TextAttr-Struktur der IntuiText-Struktur. Rückgabe: Die Länge des Strings in Pixeln.

intuitext: Adresse einer IntuiText-Struktur.

Intuition(enev)

enev: iexp

Diese Funktion übergibt eine Liste von Input-Event-Strukturen an Intuition. Nach der Dokumentation sollte diese Funktion nicht verwendet werden. Rückgabe: Adresse der ersten Struktur der Input-Event-Liste.

ItemAddress(menunr,itemnr)

menunr, itemnr: ivar

ItemAddress ermittelt die Adresse, ab der der zu suchende Menü-Eintrag im Speicher steht. Rückgabe: Adresse der MenuItem-Struktur.

menunr: Nummer des Menüs, in dem sich der Eintrag befindet.
itemnr: Nummer des Eintrags, dessen MenuItem-Adresse ermittelt werden soll.

MakeScreen(screen)

screen: iexp

Richtet den Ausgabebildschirm eines Screens ein. Keine Rückgabe.

screen: Adresse der Screen-Struktur.

ModifyIDCMP(window,flag)

window, flag: iexp

Diese Funktion ändert die IDCMP-Einstellungen des angegebenen Fensters. Keine Rückgabe.

window: Adresse der Window-Struktur.
flag: Neue IDCMP-Einstellungen.

ModifyProp(gadget,window,requester,flags,hpot,vpot, hbody,vbody)

gadget, window, requester, flags: iexp
hpot, vpot, hbody, vbody: ivar

Diese Funktion ändert die Einstellungen für ein Proportional-Gadget. Das Gadget wird nach Aufruf dieser Funktion neu gezeichnet. Keine Rückgabe.

gadget:	Adresse des Proportional-Gadgets.
window:	Adresse der Window-Struktur.
requester:	Adresse der Requester Struktur. Handelt es sich um ein Fenstergadget muß „requester" auf 0 gesetzt werden.
flags:	Neue Proportional-Flags.
hpot:	Neue horizontale Schrittweite des Sliders.
vpot:	Neue vertikale Schrittweite.
hbody:	Neue horizontale Größe.
vbody:	Neue vertikale Größe.

MoveScreen(screen,deltax,deltay)

screen: iexp
deltax, deltay: ivar

Verschiebt den Screen um den in deltax und deltay angegebenen Wert. Keine Rückgabe.

screen:	Adresse der Screen-Struktur.
deltax, deltay:	Neue Screen-Koordinate, relativ zur alten.

MoveWindow(window,deltax,deltay)

window: iexp
deltax, deltay: ivar

Verschiebt ein Fenster um den in deltax und deltay angegebenen Wert. Keine Rückgabe.

window:	Adresse der Window-Struktur.
deltax, deltay:	Neue Fensterkoordinate, relativ zur alten.

OffGadget(gadget,window,requester)

gadget, window, requester: iexp

Das angegebene Gadget wird abgeschaltet und daraufhin mit einem Punktraster überzogen um die Deaktivierung kenntlich zu machen. Diese Funktion kann nur sinnvoll auf ein Gadget angewendet werden, welches eine Imagebit-Grafik anzeigt, da anderenfalls die Rasterung nicht wieder rückgängig gemacht werden kann. Keine Rückgabe.

gadget:	Adresse der Gadget-Struktur
window:	Adresse der Window-Struktur
requester:	Adresse der Requester-Struktur. Handelt es sich um ein Fenster-Gadget kann der Wert für „requester" auf 0 gesetzt werden.

OffMenu(window,menünr)

window: iexp
menünr: ivar

Schaltet einen Menü ab, so daß es zwar noch angezeigt, aber nicht mehr angewählt werden kann. Keine Rückgabe.

window:	Adresse der Window-Struktur.
menünr:	Menünummer.

OnGadget(gadget,window,requester)

gadget, window, requester: iexp

Schaltet ein durch die Funktion OffGadget() abgeschaltetes Gadget wieder ein. Keine Rückgabe.

gadget:	Adresse der Gadget-Struktur.
window:	Adresse der Window-Struktur.
requester:	Adresse der Requester-Struktur (nur bei Requester-Gadget).

OnMenu(window,menünr)

window: iexp
menünr: ivar

Schaltet ein durch die Funktion OffMenu() abgeschaltetes Menü wieder ein. Keine Rückgabe.

window: Adresse der Window-Struktur.
menünr: Nummer des Menüs, das angeschaltet werden soll.

OpenScreen(newscreen)

newscreen: iexp

Zeigt einen neuen Screen auf dem Monitor an. Rückgabe: Adresse der Screen-Struktur oder 0.

newscreen: Adresse der NewScreen-Struktur, die die Screen-Attribute enthält.

OpenWindow(newwindow)

newwindow: iexp

Öffnet ein neues Fenter auf dem in der NewWindow-Struktur angebenen Screen. Rückgabe: Adresse der Window-Struktur.

newwindow: Adresse der NewWindow-Struktur, die die Fensterattribute enthält.

PrintIText(rastport,itext,x_off,y_off)

rastport, itext: iexp
x_off, y_off: ivar

Diese Funktion gibt die Text-Strings von IntuiText-Strukturen, in der, durch die Struktur, beschriebenen Art auf dem angegebenen Rastport aus. Ist die angegebene Struktur mit weiteren verkettet, werden alle anderen mit angezeigt. Keine Rückgabe.

rastport: Adresse der RastPort-Struktur auf dem der Text ausgebenen werden soll.
itext: Adresse der ersten IntuiText-Struktur.
x_off, y_off: Koordinate die zur Startkoordinate aller auszugebenden IntuiTexte hinzuaddiert wird.

RefreshGadgets(gadget,window,requester)

gadget,window,requester): iexp

Zeichnet die Gadget-Liste eines Fensters oder Requesters neu. Dabei wird ab dem angegebenen Gadget bis zum Ende der Liste alle Gadget neu gezeichnet. Keine Rückgabe.

gadget: Adresse der ersten Gadget-Struktur.
window: Adresse der Window-Struktur.
requester: Adresse der Requester-Struktur, wenn es sich um Requester-Gadgets handelt. Andernfallls 0.

RefreshGList(gadget,window,requester,count)

gadget, window, requester: iexp
count: ivar

Zeichnet die Gadget-Liste eines Fensters oder Requesters neu. Dabei werden nur „count" Gadgets ab dem angegebenen Gadget der Liste neu gezeichnet. Keine Rückgabe.

gadget: Adresse der ersten Gadget-Struktur.
window: Adresse der Window-Struktur.
requester: Adresse der Requester-Struktur, wenn es sich um Requester-Gadgets handelt. Andernfallls 0.
count: Anzahl der Gadgets, die neu gezeichnet werden sollen.

RefreshWindowFrame(window)

window: iexp

Erneuert die Umrandunmg eines Fensters. Keine Rückgabe.

window: Adresse der Window-Struktur.

RemakeDisplay()

Diese Funktion erneuert alle dargestellten Screen-Viewports.

RemoveGadget(window,gadget)

window, gadget: iexp

Löscht ein Gadget aus einer Fenster-Gadget-Liste.

Rückgabe: Position des Gadgets innerhalb der Liste.
window: Adresse der Window-Struktur.
gadget: Adresse der Gadget-Struktur.

RemoveGList(window,gadget,count)

window, gadget: iexp

Löscht eine „anzahl" Gadgets aus einer Fenster-Gadget-Liste. Rückgabe: Position des ersten Gadgets innerhalb der Liste.

window: Adresse der Window-Struktur.
gadget: Adresse der Gadget-Struktur.
count: Anzahl der zu löschenden Gadgets.

Request(requester,window)

requester, window: iexp

Stellt einen Requester in einem Fenster dar. Der angegebene Requester muß zuvor mit der Funktion InitRequester() initialisiert werden. Rückgabe: TRUE oder FALSE.

requester: Adresse der Requester-Struktur.
window: Adresse der Window-Struktur.

RethinkDisplay()

Diese Funktion dient zum Auffrischen der Relationen des Screens, sowie der Darstellungsliste des Coppers.

ScreenToBack(screen)

screen: iexp

Der Screen wird in den Hintergrund geschoben, so daß er von anderne Screens überdeckt ist. Keine Rückgabe.

screen: Adresse einer Screen-Struktur.

ScreenToFront(screen)

screen:iexp
Der Screen wird in der Vordergrund geholt, so daß eventuell andere angezeigte Screens hinter ihm stehen. Keine Rückgabe.

screen: Adresse einer Screen-Struktur.

SetDMRequest(window,requester)

window, requester: iexp

Stellt einen Double-Menü-Requester auf dem Screen dar. Rückgabe: TRUE oder FALSE.

window: Adresse der Window-Struktur.
requester: Adresse der Requester-Struktur.

SetMenuStrip(window,menüadr)

window, menüadr: iexp

Setzt die Menüleiste für das angegebene Fenster. Keine Rückgabe.

window: Adresse der Window-Struktur.
menüadr: Adresse der ersten Menu-Struktur.

SetPointer(window,data,höhe,breite,x,y)

window, data: iexp
höhe, breite, x, y: ivar

Setzt einen Fenster-spezifischen Mauszeiger. Keine Rückgabe.

window: Adresse der Window-Struktur.
data: Adresse der Sprite-Daten für den neuen Mauszeiger.
höhe, breite: Die Höhe und Breite des Zeigers in Pixel.
x, y: Die relative Koordinate des Aktionspunktes.

SetPrefs(pref,count,mode)

pref, count, mode: iexp

Diese Funktion speichert die System-Preferences-Einstellungen auf Diskette oder in den dafür vorgesehenen Speicherbereich.

pref: Adresse des Speicherbereichs, der die neuen Daten enthält.
count: Größe des Speicherbereichs.
mode: 0= im RAM verändern, 1= auf Diskette sichern.

SetWindowTitles(window,wtitel,stitel)

window: iexp
wtitel, stitel: sexp

Die Funktion setzt einen neuen Fenster Titel. Außerdem kann in der Funktion auch ein Screen-Titel angegeben werden, der immer dann als Screen-Titel angezeigt wird, wenn man das Fenster mit der Maus aktiviert. Keine Rückgabe.

window: Adresse der Window-Struktur.
wtitel: Neuer Fenstertitel.
stitel: Fensterspezifischer Screen-Titel.

ShiftItem(code)

code: iexp

Ermittelt die Nummer des Menü-Eintrages, der angewählt wurde. Rückgabe: Nummer des Menü-Eintrages.

code: 16-Bit-Code-Variable aus einer Message-Struktur.

ShiftMenu(code)

code: iexp

Ermittelt die Nummer des Menüs, das angewählt wurde. Rückgabe: Nummer des Menüs.

code: 16-Bit-Code-Variable aus einer Message Struktur.

ShowTitle(screen,mode)

screen: iexp
mode: ivar

Durch diese Funktion kann bestimmt werden, ob die Screen-Titelleiste im Vordergrund oder Hintergrund von Backdrop-Fenstern angezeigt werden soll.

screen:	Adresse der Screen-Struktur.
mode:	Ist 1, wenn die Titelleiste vor Backdrop-Fenstern angezeigt werden soll. Andernfalls 0.

SizeWindow(window,deltax,deltay)

window: iexp
deltax, deltay: ivar

Mit dieser Funktion kann ein Fenster vergrößert oder verkleinert werden. Keine Rückgabe.

window:	Adresse der Window-Struktur.
deltax, deltay:	Neue Größenkoordinate, relativ zur alten.

UnlockIBase(lockwert)

lockwert: iexp

Hebt den durch die Funktion LockIBase gesetzten Zugriffschutz wieder auf. Keine Rückgabe.

lockwert: Der Wert, der die Funktion LockIBase zurückliefert.

WindowLimits(window,minx,miny,maxx,maxy)

window: iexp
minx, miny, maxx, maxy: ivar

Setzt die Größenbeschränkung eines Windows neu. Rückgabe: TRUE oder FALSE

window: Adresse der Window-Struktur.
minx, miny: Kleinste Größe des Fensters.
maxx, maxy: Maximalgröße des Fensters.

WindowToBack(window)

window: iexp

Verlagert ein Fenster in den Hintergrund, so daß es von anderen Fenster teilweise oder ganz verdeckt wird. Keine Rückgabe.

window: Adresse der Window-Struktur.

WindowToFront(window)

window: iexp

Stellt das Fenster vor allen anderen dar. Keine Rückgabe.

window: Adresse der Window Struktur.

AvailFonts(buffer,number_bytes,types)

buffer, number_bytes, types: iexp

Füllt eine AvailFontsHeader-Struktur, welche die Anzahl der Fonts im Speicher oder auf Diskette enthält. Außerdem wird für jeden Zeichensatz eine AvailFont-Struktur angelegt. Rückgabe: Im Fehlerfall 0. Der Fehler kann mit IoErr() abgefragt werden.

buffer:	Adresse des Speicherbereichs für die Strukturen.
number_bytes:	Blockgröße in Bytes.
types:	AFF_MEMORY (1) oder AFF_DISK (2)

CloseFont(textFont)

textFont: iexp

Diese Funktion schließt einen zuvor mit OpenFont oder OpenDiskFont geöffneten Zeichensatz.

textFont: Adresse der TextFont-Struktur.

OpenDiskFont(textattr)

textattr: iexp

Lädt den in der TextAttr-Struktur angegebenen Zeichensatz von einer Diskette. Rückgabe: Adresse der TextFont-Struktur des Zeichensatzes.

textattr: Adresse einer TextAttr-Struktur, in der der zu ladende Zeichensatz angegeben wird.

OpenFont(textattr)

textattr: iexp

Versucht einen im Speicher abgelegten Zeichensatz (ROM-Font) in eine Text-Font-Struktur zu laden. Rückgabe: Adresse der TextFont-Struktur des Zeichensatzes.

textattr: Adresse einer TextAttr-Struktur in der der zu ladende Zeichensatz angegeben wird.

RemFont(textfont)

textfont: iexp

Löscht den angegebenen Zeichensatz aus der Zeichensatz-List, so daß er für kein Programm mehr zugänglich ist. Rückgabe: 0 wenn kein Fehler auftrat.

AttemptLockLayerRom(layer)

layer: iexp

Diese Funktion versucht einen Layer für andere Tasks zu sperren. Rückgabe: TRUE oder FALSE.

layer: Adresse der Layer-Struktur.

BeginUpdate(layer)

layer: iexp

Diese Funktion bereitet das System auf die Erneuerung eines Layers vor.

layer: Adresse der Layer-Struktur

BehindLayer(layerInfo,layer)

layerInfo, layer: iexp

Der Layer wird hinter alle anderen Layer gesetzt. Rückgabe: TRUE oder FALSE.

layerInfo: Adresse der LayerInfo-Struktur.
layer: Adresse der Layer-Struktur.

CreateBehindLayer(layerInfo,bitMap,x0,y0,x1,y1,flags,[bitMap2])

layerInfo, bitMap, [bitMap2]:iexp
x0, y0, x1, y1, flags:ivar

Erzeugt einen neuen Layer und stellt ihn hinter alle anderen Layer. Rückgabe: Adresse der Layer-Struktur.

layerInfo: Adresse der LayerInfo-Struktur.
bitMap: Adresse der BitMap-Struktur.
x0, y0: Koordinate der linken oberen Ecke.
x1, y1: Koordinate der rechten unteren Ecke.
flags: Layertyp.
[bitMap2]: Optionale Adresse einer Superbitmap.

CreateUpfrontLayer(layerInfo,bitMap,x0,y0,x1,y1,flags,[bitMap2])

layerInfo, bitMap, [bitMap2]: iexp
x0, y0, x1, y1, flags: ivar

Erzeugt einen neuen Layer und stellt ihn vor alle anderen Layer. Rückgabe: Adresse der Layer Struktur.

layerInfo:	Adresse der LayerInfo-Struktur.
bitMap:	Adresse der BitMap-Struktur.
x0, y0:	Koordinate der linken oberen Ecke.
x1, y1:	Koordinate der rechten unteren Ecke.
flags:	Layertyp.
[bitMap2]:	Optionale Adresse einer Superbitmap.

DeleteLayer(layerInfo,layer)

layerInfo, layer: iexp

Löscht einen Layer aus einer Layerliste. Keine Rückgabe.

layerInfo:	Adresse der LayerInfo-Struktur.
layer:	Adresse der Layer-Struktur.

DisposeLayerInfo(layerInfo)

layerInfo: iexp

Gibt den Speicher für die LayerInfo-Struktur frei. Keine Rückgabe.

layerInfo:	Adresse der LayerInfo-Struktur.

EndUpdate(layer,flag)

layer, flag: iexp

EndUpdate beendet das Neuzeichnen eines Simple-Refresh Layers. Diese Funktion wird auch durch die Funktion EndRefresh() aufgerufen. Keine Rückgabe.

layer:	Adresse der Layer-Struktur.
flags:	Layer-Flags

FattenLayerInfo(info)

info: iexp

Wird die Funktion InitLayer() benutzt muß mit der Funktion FattenLayerInfo zusätzlich Speicher für eine LayerInfo-Struktur reserviert werden. Keine Rückgabe.

info: Adresse einer LayerInfo Struktur.

InitLayers(layerinfo)

layerinfo: iexp

Richtet eine LayerInfo-Struktur ein. Dies ist eine Funktion des alten Betriebssystems V1.0. Der Speicher für die Struktur muß mit der Funktion AllocMem() reserviert werden. Keine Rückgabe.

layerinfo: Adresse der LayerInfo-Struktur.

LockLayer(info,layer)

info, layer: iexp

Verbietet anderen Programmen den Zugriff auf einen Layer. Keine Rückgabe.

info: Adresse der LayerInfo-Struktur.
layer: Adresse der Layer-Struktur.

LockLayerInfo(info)

info: iexp

Diese Funktion verbietet anderen Programmen den Zugriff auf die angegebene LayerInfo-Struktur. Keine Rückgabe.

info: Adresse der LayerInfo-Struktur.

LockLayerRom(layer)

layer: iexp

LockLayerRom verbietet anderen Programmen den Zugriff auf den angebenen Layer. Keine Rückgabe.

layer: Adresse der Layer-Struktur.

LockLayers(info)

Verbieten anderen Prgrammen den Zugriff auf alle Layer, die in der angegebenen LayerInfo-Struktur enthalten sind. Keine Rückgabe.

info: Adresse der LayerInfo-Struktur.

MoveLayer(info,layer,deltax,deltay)

info, layer: iexp
deltax, deltay: ivar

Mit dieser Funktion können normale Layer verschoben werden. Keine Rückgabe.

info: Adresse der LayerInfo-Struktur.
layer: Adresse der Layer-Struktur.
deltax, deltay: Neue Koordinate des Layers, relativ zur alten.

MoveLayerInFrontOf(layer1,layer2)

layer1, layer2: iexp

Mit Hilfe dieser Funktion können Layer in ihrer Lage beliebig verschoben werden. Rückgabe: TRUE oder FALSE.

layer1: Adresse des Layers der verschoben werden soll.
layer2: Adresse des Layers vor dem layer1 angezeigt werden soll.

NewLayerInfo(info)

info: iexp

Diese Funktion reserviert Speicher und intitialisiert eine LayerInfo-Struktur. Rückgabe: Adresse der neuen LayerInfo-Struktur.

info: Adresse einer bereits bestehenden LayerInfo Struktur.

ScrollLayer(info,layer,deltax,deltay)

info, layer: iexp
deltax, deltay: ivar

Diese Funktion verschiebt den Inhalt eines Super-Bitmap-Layers. Keine Rückgabe.

info: Adresse einer LayerInfo-Struktur.
layer: Adresse einer Layer-Struktur.
deltax, deltay: Neue Koordinate, relativ zur alten.

SizeLayer(info,layer,deltax,deltay)

info, layer: iexp
deltax, deltay: ivar

Mit dieser Funktion kann ein Layer vergrößert oder verkleinert werden. Keine Rückgabe.

info: Adresse der LayerInfo-Struktur.
layer: Adresse der Layer-Struktur.
deltax, deltay: Neue Größenkoordinate, relativ zur alten.

SyncSBitMap(layer)

layer: iexp

Kopiert den Inhalt eines Super-Bitmap-Layers in die Super-Bitmap. Keine Rückgabe.

layer: Adresse der Layer-Struktur.

ThinLayerInfo(info)

info: iexp

Diese Funktion löscht den zusätzlichen Speicherbereich einer LayerInfo-Struktur, der von der Funktion FattenLayerInfo() angelegt wurde. Keine Rückgabe.

info: Adresse der LayerInfo-Struktur.

UnlockLayer(layer)

layer: iexp

Diese Funktion gibt den Zugriff auf den Layer wieder frei. Keine Rückgabe.

layer: Adresse der Layer-Struktur.

UnlockLayerInfo(info)

info: iexp

Diese Funktion gibt den Zugriff auf den LayerInfo-Struktur wieder frei. Keine Rückgabe.

info: Adresse der LayerInfo-Struktur.

UnlockLayerRom(layer)

layer: iexp

Diese Funktion gibt den Zugriff auf den Layer wieder frei. Keine Rückgabe.

layer: Adresse der Layer-Struktur.

UnlockLayers(info)

info: iexp

Diese Funktion gibt den Zugriff auf alle Layer frei, die in der LayerInfo-Struktur miteinander verkettet sind. Keine Rückgabe.

info: Adresse der LayerInfo-Struktur.

Upfrontlayer(info,layer)

info, layer: iexp

Diese Funktion stellt den angegeben Layer vor allen anderen dar. Keine Rückgabe.

info: Adresse der LayerInfo-Struktur.
layer: Adresse der Layer-Struktur.

WhichLayer(info,x,y)

info: iexp
x, y: ivar

Diese Funktion ermittelt die Adresse des Layers, auf den die Koordinate zutreffen könnte. Rückgabe: Adresse der Layer-Struktur oder 0.

info: Adresse der LayerInfo-Struktur.
x, y: Koordinate innerhalb des Layers.

Close(filehandle)

filehandle: iexp

Schließt eine Ein- oder Ausgabedatei.

filehandle: Adresse der FileHandle-Struktur.

CreateDir(name)

name: svar

Erzeugt ein neues Verzeichnis unter dem angegebenen Namen, im aktuellen Verzeichnis. Ist die nicht möglich, wird eine 0 zurückgemeldet. Mit der Funktion IoErr kann dann die entsprechende Fehlernummer ermittelt werden. Bei Erfolg erhält man den „sheared read lock" für das neue Directory. Keine Rückgabe.

name: Name des Verzeichnisses.

CurrentDir(lock)

lock: iexp

CurrentDir macht ein Verzeichnis zum aktuellen Verzeichnis. Rückgabe: Adresse der Lock-Struktur des alten Verzeichnisses.

lock: Adresse der Lock-Struktur des neuen Verzeichnisses.

DateStamp(adr)

adr: iexp

Liefert Datum und Zeit im Amiga-DOS Format.

adr: Adresse von drei Langworten. Im ersten Langwort steht die Zahl für den Tag. Das zweite Langwort enthält die vergangenen Sekunden des Tages und das dritte die vergangenen Mikrosekunden der Sekunde.

Delay(time)

time: ivar

Hält den aktuellen Prozeß für die angegebene Zeit an.

time: Zeit in 1/50 Sekunden.

DeleteFile(name)

Löscht eine Datei oder Verzeichnis. Ein Verzeichnis darf dabei keine Dateien oder weitere Verzeichnisse enthalten. Rückgabe: TRUE oder FALSE. Eine Fehlernummer kann mit IoErr erfragt werden.

name: Datei oder Verzeichnisname.

DupLock(lock)

lock: iexp

Diese Funktion erzeugt eine Kopie eines „shared-read-lock" und liefert die Adresse zurück. Rückgabe: Adresse der kopierten Lock-Struktur.

lock: Adresse der Lock-Struktur.

Examine(lock,fib)

lock, fib: iexp

Die Funktion Examine schreibt die Informationen der in der Lock-Strukur angegebenen Datei oder Verzeichnisses in einen File-Info-Block. Der File-Info-Block enthält dann den Namen, die Größe, das Datum der Erstellung und den Typ der Datei oder des Verzeichnisses. Rückgabe: Bei Erfolg wird eine –1 zurückgeliefert. Andernfalls kann mit der Funktion IoErr die Fehlernummer ermittelt werden.

lock: Adresse der Lock-Struktur.
fib: Adresse der FileInfoBlock-Struktur.

Execute(anweisung,in,out)

anweisung: sexp
in, out: iexp

Mit der Funktion Execute können CLI-Befehle ausgeführt werden. Dabei muß der CLI-Befehl „Run" im C-Verzeichnis der Bootdiskette vorhanden sein. Rückgabe: TRUE oder FALSE

anweisung: Der CLI-Befehl in einem String.
in: Eingabe-File-Handle
out: Ausgabe-File-Handle

Exit(ret)

ret: ivar

Ein Programm oder Prozeß soll beendet werden. Wurde ein Programm direkt aus CLI gestartet, beendet Exit das Programm und kehrt in CLI zurück. Bei einem von der Workbench gestarteten Programm, beendet Exit den gesamten Prozeß und gibt den Speicherplatz und den Stack wieder frei.

ret: Zahl die zurückgeliefert werden soll.

ExNext(lock,fib)

lock, fib: iexp

Mit der Funktion ExNext können alle Einträge eines Verzeichnisses erfragt werden. ExNext muß dabei so oft aufgerufen werden, bis alle Einträge eines Verzeichnisses abgearbeitet sind. Die Funktion Examine() liefert den ersten Eintrag des Verzeichnisses bevor mit ExNext alle weiteren erfragt werden können. Rückgabe: Bei Erfolg –1.

lock: Adresse einer Lock-Struktur
fib: Adresse einer FileInfoBlock-Struktur

Info(lock,info)

lock, info: iexp

Info liest Informationen über alle im Gebrauch befindlichen Disketten oder Harddisks. Die Information enthält die Diskettengröße in KBytes, die Anzahl der freien und belegten Blocks und die Anzahl der Soft_Errors. Rückgabe: TRUE oder FALSE

lock: Adresse der Lock-Struktur die den Namen der Diskette enthält.
info: Adresse des Speicherbereichs, in den die Informationen kopiert werden sollen. Dieser muß unbedingt an einer durch 4 teilbaren Speicheradresse liegen.

Input()

Ermittelt das FileHandle mit der das Programm initialisiert wurde. Rückgabe: Adresse des FileHandle.

IoErr()

Ermittelt die Fehlernummer des zuletzt aufgetretenen IO-Fehlers. (siehe Fehlertabelle im Anhang) Rückgabe: Fehlernummer.

IsInteractive(handle)

handle: iexp

Diese Funktion überprüft, ob eine Datei eine Diskettendatei ist. Rückgabe: TRUE oder FALSE

LoadSeg(name)

name: svar

LoadSeg lädt ein Programm in den Speicher und zerlegt es dabei in seine Code-Segmente. Rückgabe: Adresse ab der die Segment-Liste im Speicher steht.

name: Name der Datei.

Lock(name,modus)

name: sexp
modus: iexp

Mit der Funktion Lock können Verzeichnisse oder Dateien für bestimmte Zugriffe gesperrt werden. Rückgabe: Adresse der Lock-Struktur.

name: Name der Datei oder des Verzeichnisses.
modus: Ist der Modus ACCESS_READ (–2) wird ein „shared-read-lock" erzeugt. Der Lesezugriff auf die Datei ist dann von mehreren Prgrammen aus möglich. Ist der Modus ACCESS_WRITE (–1) wird ein „write_lock" erzeugt. Der Schreibzugriff auf die Datei ist dann nur aus diesem Programm möglich.

Open(name,mode)

name: sexp
mode: iexp

Öffnet eine Datei für Schreib- oder Lesezugriffe und liefert das Filehandle zurück. Rückgabe: Filehandle.

name: Name der Datei, auf die zugeriffen werden soll.
mode: Ist „mode" gleich MODE_OLDFILE (1005), wird eine bestehende Datei zum Lesen oder Schreiben geöffnet.
Ist „mode" gleich MODE_NEWFILE (1006), wird eine neue Datei erzeugt und zum Schreiben geöffnet. Dabei wird eine eventuell bereits vorhanden Datei mit dem gleichen Namen überschrieben.

Output()

Output liefert die Adresse des Filehandle der aktuellen Ausgabe-Ebene. Rückgabe: Adresse der FileHandle-Struktur.

ParentDir(lock)

ParentDir ermittelt das Ursprungsverzeichnis eines Verzeichnisses oder einer Datei. Rückgabe: Adresse einer Lock-Struktur. Der Name des Verzeichnisses kann mit der Funktion Examine() ausgelesen werden.

lock: Adresse einer Lock-Struktur.

Read(filehandle,adr,count)

filehandle, adr, count: iexp

Liest Daten aus einer geöffneten Datei. Rückgabe: Anzahl der Bytes die tatsächlich gelesen werden konnten.

filehandle: Adresse der FileHandle-Struktur.
adr: Adresse des Speicherbereichs, in den die Daten geschrieben werden sollen.
count: Anzahl der Bytes die gelesen werden sollen.

Rename(alter,neuer)

alter, neuer:sexp

Mit dieser Funktion können Verzeichnis- oder Dateinamen umbenannt werden. Rückgabe: TRUE oder FALSE

alter: Bekannter Dateiname.
neuer: Dateiname, in den umbenannt werden soll.

Seek(filehandle,pos,mode)

filehandle, mode: iexp
pos: ivar

Mit dieser Funktion können bestimmte Positionen innerhalb von Dateien direkt angefahren werden. Rückgabe: Alte Position

filehandle: Adresse der FileHandle-Struktur der Datei.
mode: OFFSET_BEGINN (1) ab dem Anfang der Datei.
OFFSET_END (–1) ab dem Ende der Datei.
OFFSET_CURRENT (0) ab der aktuellen Position.

SetComment(name,kommentar)

name, kommentar: sexp

Schreibt einen dateispezifischen Kommentar in das Dateiverzeichis der Diskette. Rückgabe: TRUE oder FALSE

name: Name der Datei.
kammentar: Kommentartext, der maximal 80 Zeichen lang sein darf.

SetProtection(name,mode)

name: sexp
mode: ivar

Schützt eine Datei oder ein Verzeichnis. Rückgabe: TRUE oder FALSE

name: Name der Datei.
mode: 32-Bit-Wert. wovon nur 4 Bit genutzt werden können.
Bit 3 setzt den Leseschutz (nicht lesbar).
Bit 2 setzt den Schreibschutz (nicht überschreibbar).
Bit 1 setet den Ausführschutz (nicht ausführbar).
Bit 0 setzt den Löschschutz (nicht löschbar).

UnLoadSeg(segment)

segment: iexp

Löscht ein vorher mit der Funktion LoadSeg() geladenes Programm aus dem Speicher. Keine Rückgabe.

segment: Adresse der Segment-Liste.

UnLock(lock)

lock: exp

Gibt den Zugriffsschutz auf eine Datei wieder frei. Keine Rückgabe.

lock: Adresse einer Lock Struktur.

WaitForChar(filehandle,zeit)

filehandle, zeit: iexp

Überprüft, ob ein Zeichen von der spezifizierten Datei innerhalb der angegebenen Zeit empfangen werden kann. Rückgabe: TRUE oder FALSE.

filehandle: Adresse der FileHandle-Struktur.
zeit: Zeitspanne, innerhalb der das Zeichen emfangen werden soll.

Write(filehandel,daten,länge)

filehandel, daten: iexp
länge: ivar

Schreibt Daten in eine geöffnete Datei. Rückgabe: tatsächliche Anzahl geschriebener Bytes.

daten: Adresse des Speicherbereich, in dem die zu schreibenden Daten stehen.
länge: Länge des Speicherbereichs in Byte.

12 – Anhang

Raw Key Codes

| | | |

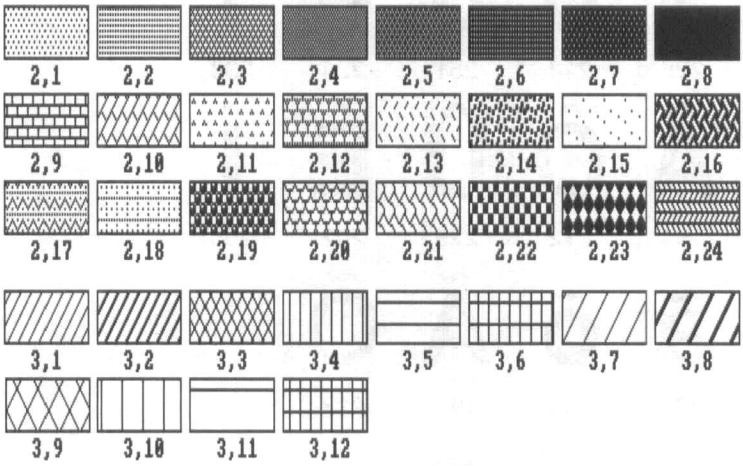
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	▒

A		¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬		®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

ä	ö	ü	ë	ï
228	246	252	235	239
Ä	Ö	Ü	Ë	Ï
196	214	220	203	207
à	ò	ù	è	ì
224	242	249	232	236
À	Ò	Ù	È	Ì
192	210	217	200	204
á	ó	ú	é	Í
225	243	250	233	237
Á	Ó	Ú	É	Í
193	211	218	201	205

â	ô	û	ê	î
226	244	251	234	238
Â	Ô	Û	Ê	Î
194	212	229	202	205
ã	õ	Ã	Õ	
227	245	195	213	
ß	ç	Ç		
223	231	199		

Füllmuster



Diese Muster können mit dem Befehl DEFFILL benutzt werden. Dabei gibt die erste Zahl den Stil (punktiert oder schraffiert) und die zweite Zahl das entsprechende Muster an.

103	Nicht genügend freier Speicher
104	Zuviele Tasks
120	Falsche Befehlszeile
121	Datei nicht ladbar
122	Fehler in einer Befehls-Bibliothek
202	Datei kann nicht beschrieben und gleichzeitig gelesen werden.
203	Datei besteht schon
204	Verzeichnis nicht gefunden
205	Datei nicht gefunden
206	Ungültige Fenster-Dimensionen
209	Ungültiger Device-Befehl
210	Dateiname zu lang oder mit ungültigen Zeichen
211	Keine gültige Lock Struktur
212	Ungültiger Typ
213	Diskerkennung nicht abgeschlossen
214	Diskette schreibgeschützt
215	Dateinamenänderung nicht gültig
216	Verzeichnis nicht leer
218	Logisches Gerät nicht auffindbar
219	Seek-Befehl mit ungültigen Parametern
220	Kommentar zu lang
221	Diskette voll
222	Datei ist vor Löschen geschützt
223	Datei ist vor Überschreiben geschützt
224	Datei ist vor Lesen geschützt
225	Keine DOS-Diskette
226	Keine Diskette im Laufwerk
232	Keine weiteren Dateien im Verzeichnis

Befehl	Kurzsyntax	Seite
!	.!	7–35
$	.$	7–49
'	.'	7–35
*	.*	2–33, 2–34
+	.+	3–10
=	.=	3–11, 3–12
==	.==	3–13
@	.@	7–24
_DosBase		10–4
_dosCmd$		10–2
_GfxBase		10–4
_IntBase		10–4
_LayersBase		10–4
{}	.{}	2–37
~	.~	2–40
–	.–	3–10
ABS()	.abs()	4–12
ABSOLUTE	.ab	2–39
ACOS()	.acos()	4–5
ADD	.add	4–2
ADD()	.add()	4–3
AFTER	.af	7–32
ALERT	.a	9–36
AND	.and	3–5
AND()	.and()	4–23
ARRAYFILL	.arr	2–12
ARRPTR()	.arrptr()	2–33
ASC()	.asc()	2–13
ASIN()	.asin()	4–5
ATN()	.atn()	4–5
BACKS	.ba	9–20
BACKW	.ba	9–32
BCHG()	.bchg()	4–19
BGET	.bg	6–12

Befehl	Kurzsyntax	Seite
BIN$()	.bin$()	2–14
BLCR()	.blcr()	4–19
BLOAD	.bl	6–12
BMOVE	.b	2–43
BOUNDARY	.bou	8–5
BOX	.bo	8–15
BPUT	.bp	6–12
BSAVE	.bs	6–12
BSET()	.bset()	4–19
BTST()	.btst()	4–19
BYTE()	.byte()	4–26
BYTE{}	.byte{}	2–37
CALL	.cal	7–54
CARD{}	.card{}	2–37
CASE	.ca	7–9
CFLOAT()	.cfloat()	2–17
CHAIN	.chai	7–43
CHAR{}	.char{}	2–37
CHDIR	.chd	6–4
CHR$()	.chr$()	2–13
CINT()	.cint()	2–17
CIRCLE	.ci	8–16
CLEAR	.cle	2–7
CLEARW	.clearw	9–32
CLIP	.cli	8–9
CLOSE	.cl	6–9
CLOSES	.closes	9–19
CLOSEW	.closew	9–26
CLR	.clr	2–7
CLS	.cls	8–20
COLOR	.c	8–2
CONT	.con	7–9
COS()	.cos()	4–5
COSQ()	.cosq()	4–6
CRSCOL	.crscol	5–13

Befehl	Kurzsyntax	Seite
CRSLIN	.crslin	5–13
CVD()	.cvd()	2–18
CVI()	.cvi()	2–18
CVL()	.cvl()	2–18
CVS()	.cvs()	2–18
DATA	.d	5–15
DATE$	.date$	2–26
DEC	.dec	4–10
DEFAULT	.defa	7–9
DEFBIT	.defbi	2–5
DEFBYT	.defb	2–5
DEFFILL	.deffill	8–3
DEFFLT	.deffl	2–5
DEFFN	.deffn	7–26
DEFINT	.defi	2–5
DEFLINE	.de	8–6
DEFNUM	.defn	5–12
DEFSTR	.defs	2–5
DEFWRD	.defw	2–5
DEG()	.deg()	4–7
DELAY	.dela	7–37
DELETE	.del	2–11
DFREE()	.dfree()	6–4
DIM	.dim	2–9
DIM?()	.dim?()	2–9
DIR	.dir	6–6
DIR$()	.dir$()	6–4
DISPLAY	.display	8–24
DIV	.div	4–2
DIV()	.div()	4–3
DO	.do	7–17
DO UNTIL	.do u	7–18
DO WHILE	.do w	7–18
DOUBLE{}	.double{}	2–37
DOWNTO	.downto	7–13

Befehl	Kurzsyntax	Seite
DPEEK()	.dpeek()	2–36
DPOKE	.dp	2–35
DRAW	.dr	8–10, 8–12
DUMP	.du	2–31
EDIT	.ed	7–38
ELLIPSE	.ell	8–16
ELSE	.e	7–4
ELSE IF bed	.e bed	7–6
END	.end	7–38
ENDFUNC	.endf	7–24
ENDIF	.en	7–4
ENDSELECT	.ends	7–9
EOF()	.eof()	6–9
EQV	.eqv	3–9
EQV()	.eqv()	4–24
ERASE	.era	2–8
ERR	.err	7–31
ERROR	.err	7–31
EVEN()	.even()	4–15
EVERY	.ev	7–32
EXEC	.exe	10–3
EXIST()	.exist()	6–7
EXIT IF	.ex	7–20
EXP()	.exp()	4–9
FALSE	.false	2–25
FIELD	.fie	6–20
FILES	.files	6–6
FILESELECT	.file	9–37
FILL	.fi	8–19
FIX()	.fix()	4–13
FLOAT{}	.float{}	2–37
FN	.fn	7–26
FOR	.f	7–13
FORM INPUT	.f input	5–6

Befehl	Kurzsyntax	Seite
RAC	.frac()	4–13
FRE()	.fre()	2–29
FRONTS	.fronts	9–20
FRONTW	.fr	9–32
FULLW	.ful	9–32
FUNCTION	.fu	7–24
GET	.ge	6–21, 8–22
GOSUB	.g	7–21
GOTO	.got	7–36
GRAPHMODE	.gr	8–7
HARDCOPY	.h	5–20
HEX$()	.hex$()	2–14
HTAB	.ht	5–13
IF	.i	7–4
IMP	.imp	3–8
IMP()	.imp()	4–24
INKEY$	.inkey$	5–2
INLINE	.inl	7–50
INP()	.inp()	6–13
INPUT	.inp	5–3, 6–15
INPUT$()	.input$()	6–14
INSERT	.ins	2–11
INSTR()	.instr()	4–30
INT()	.int()	4–13
INT{}	.int{}	2–37
KILL	.k	6–11
LEFT$()	.left$()	4–27
LEN$()	.len$()	4–34
LET	.le	2–6
LIMITW	.lim	9–33
LINE	.li	8–10

Befehl	Kurzsyntax	Seite
LINE INPUT	.li input	5–5
LINE INPUT #	.li input #	6–15
LIST	.li	7–42
LLIST	.ll	7–42
LOAD	.loa	7–40
LOC()	.loc()	6–9
LOCAL	.loc	7–23
LOCATE	.locat	5–7
LOF()	.lof()	6–9
LOG()	.log()	4–9
LOG10()	.log10()	4–9
LONG{}	.long{}	2–37
LOOP	.l	7–17
LOOP UNTIL	.l u	7–18
LOOP WHILE	.l w	7–18
LPEEK()	.lpeek()	2–36
LPOKE	.lp	2–35
LPOS()	.lpos()	5–19
LPRINT	.lpr	5–19
LSET	.ls	4–36
MALLOC()	.malloc()	2–41
MAX()	.max()	4–16
MENU	.me	9–4, 9–12
MENU KILL	.me kill	9–12
MFREE()	.mfree()	2–41
MID$()	.mid$()	4–28, 4–29
MIN()	.min()	4–16
MKD$()	.mkd$()	2–18
MKI$()	.mki$()	2–18
MKL$()	.mkl$()	2–18
MKS$()	.mks$()	2–18
MOD()	.mod()	4–3
MODE	.mod	5–11
MONITOR	.m	7–53
MOUSE	.mouse	5–17

Befehl	Kurzsyntax	Seite
MOUSEK	.mousek	5–17
MOUSEX	.mousex	5–17
MOUSEY	.mousey	5–17
MOVES	.moves	9–21
MOVEW	.mov	9–33
MUL	.mul	4–2
MUL()	.mul()	4–3
NAME	.na	6–11
NC	.inc	4–10
NEW	.new	7–39
NEXT	.n	7–13
NOT	.not	3–4
OBJECT.AX	.o.	10-13, 10-15
OBJECT.AY	.o.ay	10-13, 10-15
OBJECT.CLIP	.o.cl	10-13
OBJECT.CLOSE	.o.clo	10-13
OBJECT.ON	.o.o	10-14
OBJECT.OFF	.o.of	10-14
OBJECT.PLANES	.o.p	10-14
OBJECT.PRIORITY	.o.pr	10-14
OBJECT.SHAPE	.o.s	10-14
OBJECT.START	.o.st	10-15
OBJECT.STOP	.o.sto	10-15
OBJECT.VX	.o.v	10-15
OBJECT.VY	.o.vy	10-15
OBJECT.X	.o.x	10-15
OBJECT.Y	.o.y	10-15
OCT$()	.oct$()	2–14
ODD()	.odd()	4–15
ON BREAK CONT	.on break cont	7–28
ON BREAK GOSUB	.on break	7–28
ON COLLISION GOSUB	.on collision gosub	10-15
ON ERROR	.on error	7–29
ON ERROR GOSUB	.on error	7–29

Befehl	Kurzsyntax	Seite
ON MENU	.on menu	9–9
ON MENU BUTTON GOSUB	.on menu button	9–10
ON MENU GOSUB	.on menu gosub	9–10
ON MENU KEY GOSUB	.on menu key	9–10
ON MESSAGE GOSUB	.on message	9–10
ON x GOSUB	.on x	7–8
OPEN	.o	6–8
OPENS	.opens	9–17
OPENW	.openw	9–26
OPTION BASE	.opt base	2–10
OR	.or	3–6
OR()	.or()	4–23
OUT	.ou	6–13
PAUSE	.pa	7–37
PBOX	.pb	8–15
PCIRCLE	.pc	8–16
PEEK()	.peek()	2–36
PELLIPSE	.pe	8–16
PI	.pi	2–25
PLOT	.pl	8–10
POINT()	.point()	8–18
POKE	.po	2–35
POLYFILL	.polyf	8–17
POLYLINE	.pol	8–17
POS()	.pos()	5–13
PRED()	.pred()	4–11, 4–33
PRINT	.p	5–7
PRINT #	.p #	6–16
PRINT AT	.p at	5–7, 5–9
PRINT USING	.p using	5–9
PROCEDURE	.pro	7–21
PSAVE	.ps	7–41
PUT	.pu	6–21, 8–22

Befehl	Kurzsyntax	Seite
QSORT	.qs	2–22
QUIT	.q	7–45
RAD()	.rad()	4–7
RAND()	.rand()	4–17
RANDOM()	.random()	4–17
RANDOMIZE	.ra	4–17
RASTPORT	.ras	8–8
RCALL	.rc	7–55
READ	.rea	5–15
RECALL	.reca	6–17
RECORD	.rec	6–21
RELSEEK	.rel	6–19
REM	.r	7–35
RENAME	.ren	6–11
REPEAT	.rep	7–15
RESERVE	.rese	2–44
RESTORE	.res	5–15
RESUME NEXT	.resu next	7–29
RETURN	.ret	7–21, 7–24
RIGHT$()	.right$()	4–27
RINSTR()	.rinstr()	4–31
RND	.rnd	4–17
ROR()	.ror()	4–22
ROUND()	.round()	4–14
RSET	.rs	4–36
RUN	.ru	7–44
SAVE	.sa	7–41
SAY	.say	10–9
SCREEN()	.screen()	9–24
SEEK	.see	6–19
SELECT	.s	7–9
SETCOLOR	.set	8–2
SETDRAW	.setd	8–12
SETPEN	.sets	9–23

Befehl	Kurzsyntax	Seite
SETSTITLE	.setst	9–23
SETTIME	.sett	2–27
SETWPEN	.setwp	9–34
SETWTITLE	.setwt	9–34
SGN()	.sgn()	4–12
SHL()	.shl()	4–21
SHR()	.shr()	4–21
SIN()	.sin()	4–5
SINGLE{}	.single{}	2–37
SINQ()	.sinq()	4–6
SIZEW	.siz	9–33
SLEEP	.sle	9–9
SOUND	.so	5–22
SPACE$()	.space$()	4–32
SPC()	.spc()	4–32
SPRITE	.spr	10–5
SQR()	.sqr()	4–8
SSORT	.ss	2–22
STICK	.sti	10–7
STOP	.st	7–38
STORE	.stor	6–17
STR$()	.str$()	2–15
STRIG	.strig	10–7
STRING$()	.string$()	4–32
SUB	.sub	4–2
SUB()	.sub()	4–3
SUCC()	.succ()	4–11, 4–33
SWAP	.sw	2–20
SWAP()	.swap()	4–25
SYSTEM	.sy	7–45
TAB()	.tab()	5–13
TAN()	.tan()	4–5
TEXT	.t	8–21
TIME$	.time$	2–26
TIMER	.timer	2–28

Befehl	Kurzsyntax	Seite
TITLES	.titles	9–22
TITLEW	.ti	9–32
TOUCH	.tou	6–9
TRACE$	.trace$	7–48
TRANSLATE$()	.translate$()	10–9
TRIM$()	.trim$()	4–34
TROFF	.troff	7–46
TRON	.tr	7–46, 7–48
TRUE	.true	2–25
TRUNC()	.trunc()	4–13
TYPE()	.type()	2–30
UNTIL	.u	7–15
UPPER$()	.upper$()	4–35
V:	.v:	2–33
VAL()	.val()	2–16
VAL?()	.val?()	2–16
VARPTR()	.varptr()	2–33
VOID	.v	2–40
VSYNC	.vs	8–23
VTAB	.vt	5–13
WAVE	.wa	5–22
WEND	.we	7–16
WHILE	.w	7–16
WINDOW()	.window()	9–35
WORD()	.word()	4–26
WORD{}	.word{}	2–37
WRITE	.wr	5–7
WRITE #	.wr #	6–16
XOR	.xor	3–7
XOR()	.xor()	4–23

Betriebssystem-Funktion	Seite
AbortIO	11-3
ActivateGadget	11-58
ActivateWindow	11-58
AddAnimOb	11-24
AddBop	11-24
AddDevice	11-3
AddFont	11-24
AddFreeList	11-3
AddGadget	11-58
AddGList	11-58
AddHead	11-3
AddIntServer	11-4
AddLibrary	11-4
AddPort	11-4
AddResource	11-4
AddTail	11-5
AddTask	11-5
AddVSprite	11-24
Allocate	11-5
AllocEntry	11-5
AllocMem	11-6
AllocRaster	11-25
AllocRemember	11-59
AllocSignal	11-6
AllocTrap	11-6
AllocWBObject	11-25
AlohaWorkbench	11-59
AndRectRegion	11-25
AndRegionRegion	11-25
Animate	11-26
AreaCircle	11-26
AreaDraw	11-26
AreaEllipse	11-26
AreaEnd	11-27
AreaMove	11-27
AskFont	11-27

Betriebssystem-Funktion **Seite**

AskSoftStyle 11-27
AttemptLockLayerRom 11-78
AutoRequest 11-59
AvailFonts 11-76
AvailMem 11-6

BeginRefresh 11-60
BeginUpdate 11-78
BehindLayer 11-78
BltBitMap 11-28
BltBitMapRastPort 11-28
BltClear 11-29
BltMaskBitMapRastPort 11-29
BltPattern 11-30
BltTemplate 11-30
BuildSysRequest 11-60
BumpRevision 11-54

Cause 11-7
ChangeSprite 11-30
ClearDMRequest 11-60
ClearEOL 11-31
ClearMenuStrip 11-61
ClearPointer 11-61
ClearRectRegion 11-31
ClearRegion 11-31
ClearScreen 11-61
ClipBlit 11-31
Close 11-85
CloseDevice 11-7
CloseFont 11-76
CloseLibrary 11-7
CloseScreen 11-61
CloseWindow 11-61
CloseWorkBench 11-54
CopySBitMap 11-32

Betriebssystem-Funktion **Seite**

CreateBehindLayer 11-78
CreateDir 11-85
CreateExtIO 11-8
CreateProc 11-8
CreateStdIO 11-8
CreateUpfrontLayer 11-79
CurrentDir 11-85
CurrentTime 11-62
CWait 11-32

DateStamp 11-85
Deallocate 11-7
Debug 11-8
Delay 11-86
DeleteFile 11-86
DeleteLayer 11-79
DeviceProc 11-9
Disable 11-9
DisownBlitter 11-32
DisplayAlert 11-62
DisplayBeep 11-62
DisposeLayerInfo 11-79
DisposeRegion 11-32
DoCollision 11-33
DoIO 11-9
DoubleClick 11-62
Draw 11-33
DrawBorder 11-63
DrawCircle 11-33
DrawEllipse 11-33
DrawGList 11-34
DrawImage 11-63
DupLock 11-86

Enable 11-9
EndRefresh 11-63

Betriebssystem-Funktion **Seite**

EndRequest 11-64
EndUpdate 11-79
Enqueue 11-9
Examine 11-86
Execute 11-87
Exit 11-87
ExNext 11-87

FattenLayerInfo 11-80
FindName 11-10
FindPort 11-10
FindResident 11-10
FindTask 11-10
FindToolTypes 11-54
Flood 11-34
Forbid 11-11
FreeColorMap 11-34
FreeCopList 11-35
FreeCprList 11-35
FreeDiskObject 11-54
FreeEntry 11-11
FreeFreeList 11-11
FreeGBuffers 11-35
FreeMem 11-11
FreeRaster 11-11
FreeRemember 11-64
FreeSignal 11-12
FreeSprite 11-35
FreeSysRequest 11-64
FreeTrap 11-12
FreeVPortCopLists 11-36
FreeWBObject 11-55

GetCC 11-12
GetColorMap 11-36
GetDefPrefs 11-65

Betriebssystem-Funktion **Seite**

GetDiskObject 11-55
GetGBuffers 11-36
GetIcon 11-55
GetMsg 11-12
GetPrefs 11-65
GetRGB4 11-36
GetScreenData 11-37
GetSprite 11-37
GetWBObject 11-55

Info 11-88
InitArea 11-38
InitBitMap 11-38
InitCode 11-13
InitGels 11-38
InitGMasks 11-39
InitLayers 11-80
InitMasks 11-39
InitRastPort 11-39
InitRequester 11-65
InitResident 11-13
InitStruct 11-13
InitTmpRas 11-39
InitView 11-40
InitVPort 11-40
Input 11-88
Insert 11-13
InstallClipRegion 11-40
IntuiTextLength 11-65
Intuition 11-66
IoErr 11-88
IsInteractive 11-88
ItemAddress 11-66

LoadRGB4 11-40
LoadSeg 11-88

Betriebssystem-Funktion **Seite**

LoadView 11-41
Lock 11-89
LockIBase 11-41
LockLayer 11-80
LockLayerInfo 11-80
LockLayerRom 11-81
LockLayers 11-81

MakeLibrary 11-14
MakeScreen 11-66
MakeVPort 11-41
MatchToolValue 11-56
MenuNum 11-42
ModifyIDCMP 11-66
ModifyProp 11-67
Move 11-42
MoveLayer 11-81
MoveLayerInFrontOf 11-81
MoveScreen 11-67
MoveSprite 11-42
MoveWindow 11-67
MrgCop 11-42

NewLayerInfo 11-82
NewRegion 11-43

OffGadget 11-68
OffMenu 11-68
OnGadget 11-68
OnMenu 11-69
Open 11-89
OpenDevice 11-14
OpenDiskFont 11-76
OpenFont 11-77
OpenLibrary 11-15
OpenResource 11-15

Betriebssystem-Funktion **Seite**

OpenScreen 11-69
OpenWindow 11-69
OpenWorkBench 11-56
OrRectRegion 11-43
OrRegionRegion 11-43
Output 11-89
OwnBlitter 11-43

ParentDir 11-90
Permit 11-15
PolyDraw 11-43
PrintIText 11-69
Procure 11-15
PutDiskObject 11-56
PutIcon 11-56
PutMsg 11-16
PutWBObject 11-57

QBlit 11-44
QBSBlit 11-44

RasSize 11-44
RawDoFmt 11-16
RawKeyConvert 11-16
Read 11-90
ReadPixel 11-44
RectFill 11-45
RefreshGadgets 11-70
RefreshGList 11-70
RefreshWindowFrame 11-71
RemakeDisplay 11-71
RemDevice 11-17
RemFont 11-77
RemHead 11-17
RemIBob 11-45
RemLibrary 11-17

Betriebssystem-Funktion **Seite**

Remove 11-17
RemoveGadget 11-71
RemoveGList 11-71
RemPort 11-18
RemResource 11-18
RemTail 11-18
RemTask 11-18
RemVSprite 11-45
Rename 11-90
ReplyMsg 11-19
ReportMouse 11-46
Request 11-71
RethinkDisplay 11-72

ScreenToBack 11-72
ScreenToFront 11-72
ScrollLayer 11-82
ScrollRaster 11-46
ScrollVPort 11-46
Seek 11-90
SendIO 11-19
SetAfPt 11-46
SetAPen 11-47
SetBPen 11-47
SetCollision 11-47
SetComment 11-91
SetDMRequest 11-72
SetDrMd 11-48
SetDrPt 11-48
SetExept 11-19
SetFont 11-48
SetFunction 11-19
SetIntVector 11-20
SetMenuStrip 11-72
SetOPen 11-48
SetPointer 11-73

Betriebssystem-Funktion **Seite**

SetPrefs 11-73
SetProtection 11-91
SetRast 11-49
SetRGB4 11-49
SetRGB4CM 11-49
SetSignal 11-20
SetSoftStyle 11-49
SetSr 11-20
SetTaskPri 11-20
SetWindowTitles 11-73
SetWrMsk 11-21
ShiftItem 11-74
ShiftMenu 11-74
ShowTitle 11-74
Signal 11-21
SizeLayer 11-82
SizeWindow 11-74
SortGList 11-50
SumLibrary 11-21
SuperState 11-21
SwapBitsRastPortClipRect 11-21
SyncSBitMap 11-82

Text 11-50
TextLength 11-50
ThinLayerInfo 11-83
Translate 11-51

UCopperListInit 11-51
UnLoadSeg 11-91
UnLock 11-92
UnlockIBase 11-75
UnlockLayer 11-83
UnlockLayerInfo 11-83
UnlockLayerRom 11-83
UnlockLayers 11-84

Übersicht Librarys

Betriebssystem-Funktion	Seite
Upfrontlayer	11-84
UserState	11-22
Vacate	11-22
VBeamPos	11-51
ViewAddress	11-51
ViewPortAddress	11-52
Wait	11-22
WaitBlit	11-52
WaitBOVP	11-52
WaitForChar	11-92
WaitIO	11-22
WaitPort	11-23
WBenchToBack	11-57
WBenchToFront	11-57
WhichLayer	11-84
WindowLimits	11-75
WindowToBack	11-75
WindowToFront	11-75
Write	11-92
WritePixel	11-52
XorRectRegion	11-52
XorRegionRegion	11-53

13 - Lineare Operationen mit Vektoren und Matrizen

Alle in diesem Kapitel beschriebenen Funktionen beziehen sich nur auf ein- und/oder zweidimensionale Felder mit Fließkommavariablen.

Systembefehle

MAT BASE 0
MAT BASE 1

Der Befehl MAT BASE ist nur dann sinnvoll, wenn im Programm OPTION BASE 0 gilt. In diesem Falle kann mit MAT BASE 1 der Offset für den Beginn der Zeilen- und Spaltenindizierung von ein- oder zweidimensionalen Feldern mit Fließkommavariablen für die Matrizenoperationen auf 1 gesetzt werden. MAT BASE 0 bewirkt nach einem MAT BASE 1 die Rücksetzung dieses Offsets auf 0.

Kürzel: m b 0 bzw. m b 1

Die durch MAT BASE n vorgenommene Einstellung wirkt sich aus auf die Kommandos

MAT READ
MAT PRINT
MAT CPY
MAT XCPY
MAT ADD
MAT SUB und
MAT MUL

Voreingestellt ist MAT BASE 1.

Beispiel:

```
OPTION BASE 0
MAT BASE 1
DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
DIM a(3,3)
MAT READ a()
PRINT a(1,1)
```

--> Gibt den Wert 1 aus.

Erzeugungsbefehle

MAT CLR a()
MAT SET a() = x
MAT ONE a()

```
a:  Name eines Feldes mit numerischen Variablen
x:  aexp
```

MAT CLR a() entspricht einem ARRAYFILL a(),0, d.h. der Befehl setzt alle Elemente des Feldes (Matrix oder Vektor) a() auf den Wert 0.

Kürzel: m cl a()

MAT SET a() = x entspricht einem ARRAYFILL a(),x, d.h. der Befehl setzt alle Elemente des Feldes a() (Matrix oder Vektor) auf den Wert x.

Kürzel: m se a() = x

MAT ONE a() erzeugt aus einer quadratischen Matrix a() eine Einheitsmatrix, d.h. eine quadratische Matrix, in der die Elemente a(1,1), a(2,2),...,a(n,n) identisch gleich 1 und alle anderen Elemente identisch gleich 0 sind.

Kürzel: m o a()

Beispiele:

```
DATA 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
DIM a(3,3)
MAT READ a()
PRINT a(1,1)
MAT CLR a()
PRINT a(1,1)

--> gibt zunächst den Wert 1, dann den Wert 0 aus.
```

```
DIM a(5,7)
FOR i%=1 TO 5
  FOR j%=1 TO 7
    a(i%,j%)=RAND(10)
  NEXT j%
NEXT i%
MAT SET a(),5.3
FOR i%=1 TO 5
  FOR j%=1 TO 7
    PRINT a(i%,j%)
  NEXT j%
NEXT i%
```

--> Gibt 35 mal den Wert 5.3 aus.

```
DIM a(3,3)
MAT ONE a()
MAT PRINT a()
```

```
--> Gibt  1,0,0
          0,1,0
          0,0,1   aus.
```

Schreib- und Lesebefehle

MAT READ a()
MAT PRINT [#i,]a()[,g,n]
MAT INPUT #i,a()

```
i,g,n:   iexp
a:       Name eines Feldes mit numerischen Variablen
```

MAT READ a() liest eine vorher dimensionierte Matrix bzw. einen Vektor aus DATAzeilen ein.

Kürzel: m r a()

Beispiel:

```
DATA 1,2,3,4,5,6,7,8,9,10
DIM a(2,5)
MAT READ a()
PRINT a(2,4)

--> gibt den Wert 9 aus.
```

MAT PRINT [#i,]a()[,g,n] gibt eine Matrix oder einen Vektor aus. Vektoren werden dabei in einer Zeile ausgegeben, mit Trennung der Elemente durch Kommata. Bei Matrizen wird nach jeder Zeile ein Zeilenvorschub ausgeführt.

Optional kann, wie bei PRINT, mit #i die Ausgabe umgeleitet werden.

Bei Angabe von g und n erfolgt eine Formatierung der Zahlen wie bei STR$(x,g,n).

Kürzel: m p [#i,]a()[g,n] oder m ? [#i,]a()[g,n]

Beispiel:

```
DATA 1,2.33333,3
DATA 7,5.25873,9.376
DATA 3.23,7.2,8.999
DIM a(3,3)
MAT READ a()
MAT PRINT a()
PRINT"---------------"
MAT PRINT a(),7,3

--> ergibt: 1,2.33333,3
            7,5.25873,9.376
            3.23,7.2,8.999
            ---------------
            1.000,2.333,3.000
            7.000,5.259,9.376
            3.230,7.200,8.999
```

MAT INPUT #1,a() liest eine Matrix oder einen Vektor in ASCII aus einer Datei ein (Format umgekehrt zu MAT PRINT, Kommata und Zeilenvorschübe dürfen variiert werden, wie bei INPUT #).

Kürzel: m i #i,a()

Beispiel:

```
OPEN "o",#1,"Test.DAT"
DIM a(3,3)
MAT ONE a()
MAT PRINT #1,a()
CLOSE #1
MAT CLR a()
```

```
OPEN "i",#1,"Test.DAT"
MAT INPUT #1,a()
CLOSE #1
MAT PRINT a()

--> Gibt  1,0,0
          0,1,0
          0,0,1   aus.
```

Kopier- und Vertauschungsbefehle

MAT CPY a([i,j]) = b([k,l])[,h,w]
MAT XCPY a([i,j]) = b([k,l])[,h,w]
MAT TRANS a()[= b()]

a,b: Name von Feldern mit numerischen Variablen i,j,k,l,h,w: iexp

MAT CPY a([i,j]) = b([k,l])[,h,w] kopiert ab dem durch l,k beschriebenen Zeilen- bzw. Spaltenoffset der Matrix b h Zeilen mit jeweils w Elementen der Matrix b an den durch i,j beschriebenen Zeilen- bzw. Spaltenoffset der Matrix a.

Kürzel: m c a(i,j) = b(k,l),h,w,

m x a(i,j) = b(k,l),h,w,

m t a() = b().

Beispiel:

```
DIM a(5,5),b(4,4)
MAT SET a()=1
FOR i%=1 TO 4
  FOR j%=1 TO 4
    b(i%,j%)=SUCC(i%)
  NEXT j%
NEXT i%
'
MAT PRINT a()
PRINT "-------------"
MAT PRINT b()
PRINT "-------------"
MAT CPY a(2,2)=b(2,2),3,3
MAT PRINT a()
```

```
--> Gibt      1,1,1,1,1
              1,1,1,1,1
              1,1,1,1,1
              1,1,1,1,1
              1,1,1,1,1
              -------------
              2,2,2,2
              3,3,3,3
              4,4,4,4
              5,5,5,5
              -------------
              1,1,1,1,1
              1,3,3,3,1
              1,4,4,4,1
              1,5,5,5,1
              1,1,1,1,1    aus.
```

Spezialfälle

MAT COPY a()=b() kopiert bei Matrizen gleicher Ordnung die komplette Matrix b in die Matrix a.

Dabei werden nur die Elemtente kopiert, für die sowohl in der Quell- als auch in der Zielmatrix identische Indizes vorliegen.

Kürzel: m c a()=b()

Beispiel:

```
DIM a(5,3),b(4,4)
MAT SET a()=1
FOR i%=1 TO 4
  FOR j%=1 TO 4
    b(i%,j%)=SUCC(i%)
  NEXT j%
NEXT i%
'
MAT PRINT a()
PRINT "-------------"
MAT PRINT b()
PRINT "-------------"
MAT CPY a()=b(),3,3
MAT PRINT a()
```

```
--> Gibt      1,1,1
              1,1,1
              1,1,1
              1,1,1
              1,1,1
              -------------
              2,2,2,2
              3,3,3,3
              4,4,4,4
              5,5,5,5
```

```
-------------
2,2,2
3,3,3
4,4,4
1,1,1
1,1,1     aus.
```

MAT COPY a(i,j) = b() kopiert die Matrix b ab dem durch MAT BASE festgelegten Zeilen- bzw. Spaltenoffset an den durch i,j beschriebenen Zeilen- bzw. Spaltenoffset der Matrix a. Dabei werden nur die Elemente kopiert, für die sowohl in der Quell- als auch in der Zielmatrix identische Indizes vorliegen.

Kürzel: m c a(i,j) = b()

Beispiel:

```
DIM a(5,3),b(4,4)
MAT SET a()=1
FOR i%=1 TO 4
  FOR j%=1 TO 4
    b(i%,j%)=SUCC(i%)
  NEXT j%
NEXT i%
'
MAT PRINT a()
PRINT "-------------"
MAT PRINT b()
PRINT "-------------"
MAT CPY a(2,2)=b(2,2),3,3
MAT PRINT a()

--> Gibt      1,1,1
              1,1,1
              1,1,1
              1,1,1
              1,1,1
```

```
-------------
2,2,2,2
3,3,3,3
4,4,4,4
5,5,5,5
-------------
1,1,1
1,3,3
1,4,4
1,5,5
1,1,1     aus.
```

MAT COPY a() = b(i,j) kopiert die Matrix b ab dem durch i,j beschriebenen Zeilen- bzw. Spaltenoffset an den durch MAT BASE festgelegten Offset der Matrix a. Dabei werden nur die Elemtente kopiert, für die sowohl in der Quell- als auch in der Zielmatrix identische Indizes vorliegen.

Kürzel: m c a() = b(i,j)

Beispiel:

```
DIM a(5,3),b(4,4)
MAT SET a()=1
FOR i%=1 TO 4
  FOR j%=1 TO 4
    b(i%,j%)=SUCC(i%)
  NEXT j%
NEXT i%
'
MAT PRINT a()
PRINT "-------------"
MAT PRINT b()
PRINT "-------------"
MAT CPY a()=b(2,2),3,3
MAT PRINT a()
```

```
--> Gibt      1,1,1
              1,1,1
              1,1,1
              1,1,1
              1,1,1
              -------------
              2,2,2,2
              3,3,3,3
              4,4,4,4
              5,5,5,5
              -------------
              3,3,3
              4,4,4
              5,5,5
              1,1,1
              1,1,1    aus.
```

MAT COPY a(i,j) = b(k,l) kopiert die Matrix b ab dem durch k,l beschriebenen Zeilen- bzw. Spaltenoffset an den Offset i,j der Matrix a. Dabei werden nur die Elemtente kopiert, für die sowohl in der Quell- als auch in der Zielmatrix identische Indizes vorliegen.

Kürzel: m c a(i,j) = b(k,l)

Beispiel:

```
DIM a(5,3),b(4,4)
MAT SET a()=1
FOR i%=1 TO 4
  FOR j%=1 TO 4
    b(i%,j%)=SUCC(j%)
  NEXT j%
NEXT i%
'
MAT PRINT a()
PRINT "-------------"
MAT PRINT b()
```

```
PRINT "-------------"
MAT CPY a(2,2)=b(2,2)
MAT PRINT a()

--> Gibt      1,1,1
              1,1,1
              1,1,1
              1,1,1
              1,1,1
              -------------
              2,3,4,5
              2,3,4,5
              2,3,4,5
              2,3,4,5
              -------------
              1,1,1
              1,3,4
              1,3,4
              1,3,4
              1,1,1    aus.
```

MAT COPY a() = b(),h,w kopiert ab dem durch MAT BASE festgelegten Zeilen- bzw. Spaltenoffset h Zeilen mit jeweils w Elementen der Matrix b an den durch MAT BASE festgelegten Zeilen- bzw. Spaltenoffset der Matrix a. Dabei werden nur die Elemente kopiert, für die sowohl in der Quell- als auch in der Zielmatrix identische Indizes vorliegen.

Kürzel: m c a() = b()

Beispiel:

```
DIM a(5,3),b(4,4)
MAT SET a()=1
FOR i%=1 TO 4
  FOR j%=1 TO 4
    b(i%,j%)=SUCC(j%)
  NEXT j%
```

```
NEXT i%
'
MAT PRINT a()
PRINT "-------------"
MAT PRINT b()
PRINT "-------------"
MAT CPY a()=b()
MAT PRINT a()
```

```
--> Gibt      1,1,1
              1,1,1
              1,1,1
              1,1,1
              1,1,1
              -------------
              2,3,4,5
              2,3,4,5
              2,3,4,5
              2,3,4,5
              -------------
              2,3,4
              2,3,4
              2,3,4
              2,3,4
              1,1,1    aus.
```

MAT XCPY a([i,j]) = b([k,l])[,h,w] funktioniert prinzipiell wie MAT CPY a([i,j]) = b([k,l])[,h,w], nur daß hier beim Kopieren der Matrix b in die Matrix a eine Transponierung der Matrix b vorgenommen wird, d.h. Zeilen und Spalten der Matrix b werden beim Kopieren in die Matrix a vertauscht. Die Matrix b bleibt jedoch unverändert. Dabei werden nur die Elemente kopiert, für die sowohl in der Quell- als auch in der Zielmatrix identische Indizes vorliegen.

Kürzel: m x a(i,j) = b(k,l),h,w

Beispiel:

```
DIM a(5,3),b(4,4)
MAT SET a()=1
FOR i%=1 TO 4
  FOR j%=1 TO 4
    b(i%,j%)=SUCC(j%)
  NEXT j%
NEXT i%
'
MAT PRINT a()
PRINT "-------------"
MAT PRINT b()
PRINT "-------------"
MAT XCPY a(2,2)=b(2,2),3,3
MAT PRINT a()
```

```
--> Gibt      1,1,1
              1,1,1
              1,1,1
              1,1,1
              1,1,1
              -------------
              2,3,4,5
              2,3,4,5
              2,3,4,5
              2,3,4,5
              -------------
              1,1,1
              1,3,3
              1,4,4
              1,5,5
              1,1,1     aus.
```

Spezialfälle

Wie bei MAT CPY a(i,j) = b(k,l),w,h.

Wird MAT CPY bzw. MAT XCPY auf Vektoren angewandt, so werden ggf. j und l nicht beachtet. Nach einem DIM a(n),b(m) werden a() und b() als Zeilenvektoren aufgefaßt, d.h. als Matrizen vom Typ (1,n) bzw. (1,m).

Um a und b als Spaltenvektoren zu behandeln, müssen sie als Matrizen vom Typ (n,1) bzw. (m,1) dimensioniert werden, d.h. DIM a(n,1),b(n,1).

Falls beide Vektoren die gleiche Ordnung besitzen (beide sind Zeilen- bzw. Spaltenvektoren) muß MAT CPY verwendet werden. Unabhängig vom Typ der Vektoren a und b faßt MAT CPY beide Vektoren syntaxmäßig immer als Spaltenvektoren auf, so daß für die korrekte Syntax von MAT CPY immer MAT CPY a(n,1) = b(m,1) angegeben werden muß!

Beispiel:

```
DIM a(10),b(5)       ' a() und b() sind Zeilenvektoren
MAT SET a()=1
FOR i%=1 TO 5
  b(i%)=SUCC(i%)
NEXT i%
PRINT "a():   ";
MAT PRINT a()
PRINT "b():   ";
MAT PRINT b()
PRINT STRING$(45,"-")
```

MAT CPY a(3,1) = b(1,1) ! faßt a() und b() syntaxmäßig als Spaltenvektoren auf.

```
PRINT "MAT CPY a(3,1)=b(1,1):  ";
MAT PRINT a()
```

```
--> Gibt    a():  1,1,1,1,1,1,1,1,1,1
            b():  2,3,4,5,6
            ----------------------------------------
            MAT CPY a(3,1)=b(1,1):  1,1,2,3,4,5,6,1,1,1      aus.
```

Für MAT XCPY muß einer der beiden Vektoren a und b explizit als Zeilenvektor, der andere explizit als Spaltenvektor dimensioniert sein: z.B. DIM a(1,10),b(5,1), da MAT XCPY den zweiten Vektor erst transponiert und dann in den ersten Vektor kopiert. MAT XCPY kann somit nur für die Fälle DIM a(1,n),b(m,1): a() = Zeilenvektor, b() = Spaltenvektor und DIM a(n,1),b(1,m): a() = Spaltenvektor, b() = Zeilenvektor verwendet werden.

Beispiel:

```
DIM a(1,10),b(5,1)
MAT SET a()=1
FOR i%=1 TO 5
  b(i%,1)=SUCC(i%)
NEXT i%
MAT PRINT a()
PRINT
MAT PRINT b()
MAT XCPY a(1,3)=b(1,1)
PRINT
MAT PRINT a()
```

```
--> Gibt   1,1,1,1,1,1,1,1,1,1

           2
           3
           4
           5
           6

           1,1,2,3,4,5,6,1,1,1  aus.
```

Optional können auch beim Kopieren von Vektoren mit MAT CPY bzw. MAT XCPY die Parameter h und w verwendet werden. Dabei gilt jedoch: Bei MAT CPY wird für w => 1 nur der Parameter h beachtet. Für w=0 findet kein Kopieren statt.

Bei MAT XCPY wird für w=>1 nur h beachtet, falls b ein Spaltenvektor ist, der nach der Transponierung in einen Zeilenvektor kopiert werden soll. Für w=0 findet kein Kopieren statt. Andererseits wird für h=>1 nur w beachtet, falls b ein Zeilenvektor ist, der nach der Transponierung in einen Spaltenvektor kopiert werden soll. Hier findet für h=0 kein Kopieren statt.

MAT TRANS a()=b() kopiert die Transponierte der Matrix b in die Matrix a, falls a und b entsprechend dimensioniert sind, d.h. die Zeilenzahl von a muß der Spaltenzahl von b, und die Spaltenzahl von a muß der Zeilenzahl von n entsprechen: z.B. DIM a(n,m),b(m,n)

Beispiel:

```
DIM a(3,4),b(4,3)
MAT SET b()=4
MAT SET a()=1
MAT PRINT a()
PRINT STRING$(10,"-")
MAT PRINT b()
PRINT STRING$(10,"-")
MAT TRANS a()=b()
MAT PRINT a()

--> gibt    1,1,1,1
            1,1,1,1
            1,1,1,1
            ----------
            4,4,4
            4,4,4
            4,4,4
            4,4,4
```

```
          ----------
          4,4,4,4
          4,4,4,4
          4,4,4,4          aus.
```

Liegt eine quadratische Matrix, d.h. eine Matrix mit gleicher Zeilen- und Spaltenzahl vor, so kann MAT TRANS a() verwendet werden. Dieser Befehl vertauscht die Zeilen und Spalten der Matrix a und schreibt die so veränderte Matrix wieder in a zurück. Dabei geht die Ursprungsmatrix a verloren (kann jedoch durch ein erneutes MAT TRANS a() wiederhergestellt werden).

Kürzel: m t a()

Beispiel:

```
DIM a(5,5)
FOR i%=1 TO 5
  FOR j%=1 TO 5
    a(i%,j%)=j%
  NEXT j%
NEXT i%
MAT PRINT a()
PRINT STRING$(10,"-")
MAT TRANS a()
MAT PRINT a()

--> Gibt    1,2,3,4,5
            1,2,3,4,5
            1,2,3,4,5
            1,2,3,4,5
            1,2,3,4,5
            ----------
```

```
1,1,1,1,1
2,2,2,2,2
3,3,3,3,3
4,4,4,4,4
5,5,5,5,5     aus.
```

Operationsbefehle

MAT ADD a()=b()+c()
MAT ADD a(),b()
MAT ADD a(),x

MAT SUB a()=b()-c()
MAT SUB a(),b()
MAT SUB a(),x

MAT MUL a()=b()*c()
MAT MUL x=a()*b()
MAT MUL x=a()*b()*c()
MAT MUL a(),x

MAT NORM a(),0
MAT NORM a(),1

MAT DET x=a([i,j])[,n]
MAT QDET x=a([i,j])[,n]
MAT RANG x=a([i,j])[,n]
MAT INV a()=b()

```
a,b,c:  Namen von numerischen Fließkommafeldern
x:      aexp; skalare Größe
i,j,n:  aexp
```

MAT ADD a()=b()+c() ist nur für Matrizen (Vektoren) gleicher Ordnung definiert, z.B. DIM a(n,m),b(n,m),c(n,m) oder DIM a(n),b(n),c(n). Die Matrix c wird elementweise zur Matrix b addiert und das Ergebnis in die Matrix a geschrieben.

Kürzel: m a()=b()+c()

Beipiel:

```
DIM a(3,5),b(3,5),c(3,5)
MAT SET b()=3
MAT SET c()=4
MAT PRINT b()
PRINT STRING$(10,"-")
MAT PRINT c()
PRINT STRING$(10,"-")
MAT ADD a()=b()+c()
MAT PRINT a()
```

```
--> Gibt   3,3,3,3,3
           3,3,3,3,3
           3,3,3,3,3
           ----------
           4,4,4,4,4
           4,4,4,4,4
           4,4,4,4,4
           ----------
           7,7,7,7,7
           7,7,7,7,7
           7,7,7,7,7    aus.
```

MAT ADD a(),b() ist nur für Matrizen (Vektoren) gleicher Ordnung definiert, z.B. DIM a(n,m),b(n,m) oder DIM a(n),b(n). Die Matrix b wird elementweise zur Matrix a addiert und das Ergebnis in die Matrix a geschrieben. Die ursprüngliche Matrix a geht dabei verloren.

Kürzel: m a a(),b()

Beispiel:

```
DIM a(3,5),b(3,5)
MAT SET a()=1
MAT SET b()=3
MAT PRINT a()
```

```
PRINT STRING$(10,"-")
MAT PRINT b()
PRINT STRING$(10,"-")
MAT ADD a(),b()
MAT PRINT a()

--> Gibt    1,1,1,1,1
            1,1,1,1,1
            1,1,1,1,1
            ----------
            3,3,3,3,3
            3,3,3,3,3
            3,3,3,3,3
            ----------
            4,4,4,4,4
            4,4,4,4,4
            4,4,4,4,4       aus.
```

MAT ADD a(),x ist für alle Matrizen (Vektoren) definiert. Dabei wird der Skalar x elementweise zur Matrix a addiert und das Ergebnis in die Matrix a geschrieben. Die ursprüngliche Matrix a geht dabei verloren.

Kürzel: m a a(),x

Beispiel:

```
DIM a(3,5)
MAT SET a()=1
MAT PRINT a()
PRINT STRING$(10,"-")
MAT ADD a(),5
MAT PRINT a()

--> gibt    1,1,1,1,1
            1,1,1,1,1
            1,1,1,1,1
            ----------
```

```
6,6,6,6,6
6,6,6,6,6
6,6,6,6,6     aus.
```

MAT SUB a()=b()+c() ist nur für Matrizen (Vektoren) gleicher Ordnung definiert, z.B. DIM a(n,m),b(n,m),c(n,m) oder DIM a(n),b(n),c(n). Die Matrix c wird elementweise von der Matrix b subtrahiert und das Ergebnis in die Matrix a geschrieben.

Kürzel: m a()=b()-c()

Beipiel:

```
DIM a(3,5),b(3,5),c(3,5)
MAT SET b()=5
MAT SET c()=3
MAT PRINT b()
PRINT STRING$(10,"-")
MAT PRINT c()
PRINT STRING$(10,"-")
MAT SUB a()=b()-c()
MAT PRINT a()

--> Gibt   5,5,5,5,5
           5,5,5,5,5
           5,5,5,5,5
           ----------
           3,3,3,3,3
           3,3,3,3,3
           3,3,3,3,3
           ----------
           2,2,2,2,2
           2,2,2,2,2
           2,2,2,2,2   aus.
```

MAT SUB a(),b() ist nur für Matrizen (Vektoren) gleicher Ordnung definiert, z.B. DIM a(n,m),b(n,m) oder DIM a(n),b(n). Die Matrix b wird

elementweise von der Matix a subtrahiert und das Ergebnis in die Matrix a geschrieben. Die ursprüngliche Matrix a geht dabei verloren.

Kürzel: m s a(),b()

Beispiel:

```
DIM a(3,5),b(3,5)
MAT SET a()=3
MAT SET b()=1
MAT PRINT a()
PRINT STRING$(10,"-")
MAT PRINT b()
PRINT STRING$(10,"-")
MAT SUB a(),b()
MAT PRINT a()

--> Gibt    3,3,3,3,3
            3,3,3,3,3
            3,3,3,3,3
            ----------
            1,1,1,1,1
            1,1,1,1,1
            1,1,1,1,1
            ----------
            2,2,2,2,2
            2,2,2,2,2
            2,2,2,2,2   aus.
```

MAT SUB a(),x ist für alle Matrizen (Vektoren) definiert. Dabei wird der Skalar x elementweise von Matrix a subtrahiert und das Ergebnis in die Matrix a geschrieben. Die ursprüngliche Matrix a geht dabei verloren.

Kürzel: m s a(),x

Beispiel:

```
DIM a(3,5)
MAT SET a()=6
MAT PRINT a()
PRINT STRING$(10,"-")
MAT SUB a(),5
MAT PRINT a()

--> gibt   6,6,6,6,6
           6,6,6,6,6
           6,6,6,6,6
           ----------
           5,5,5,5,5
           5,5,5,5,5
           5,5,5,5,5
           ----------
           1,1,1,1,1
           1,1,1,1,1
           1,1,1,1,1   aus.
```

MAT MUL a() = b()*c() ist für Matrizen 'geeigneter' Ordnung definiert. Multipliziert werden die Matrizen b und c. Das Ergebnis dieses Matrizenproduktes wird in die Matrix a geschrieben. Damit das Ergebnis eines Matrizenproduktes definiert ist, muß die linksstehende Matrix des Produktes (in diesem Falle die Matrix b) genausoviele Spalten aufweisen wie die rechtsstehende Matrix des Produktes (in diesem Falle c) Zeilen besitzt. Die Matrix a muß in diesem Falle genausoviele Zeilen wie b und genausoviele Spalten wie c aufweisen, z.B.

```
DIM a(2,2),b(2,3),c(3,2)
```

Matrizen werden nach der Beziehung 'Zeile mal Spalte' multipliziert. D.h. man erhält das Element a(i,j), indem man die Elemente der i-ten Zeile der Matrix b mit den Elementen der j-ten Spalte der Matrix c elementweise multipliziert und die einzelnen Produkte aufaddiert.

Kürzel: m a() = b()*c()

Beispiel:

```
DIM a(2,2),b(2,3),c(3,2)
MAT SET b()=1
DATA 1,2,-3,4,5,-1
MAT READ c()
MAT PRINT b(),5,1
PRINT STRING$(18,"-")
MAT PRINT c(),5,1
PRINT STRING$(18,"-")
MAT MUL a()=b()*c()
MAT PRINT a(),5,1
```

```
--> Gibt      1.0,  1.0,  1.0
              1.0,  1.0,  1.0
            ----------------
              1.0,  2.0
             -3.0,  4.0
              5.0, -1.0
            ----------------
              3.0,  5.0
              3.0,  5.0          aus.
```

Liegen anstelle von Matrizen Vektoren vor, so ergibt MAT MUL a() = b()*c() das dyadische (sog. äußere) Produkt zweier Vektoren.

Beispiel:

```
DIM a(3,3),b(3),c(3)
DATA 1,2,-3,4,5,-1
MAT READ b()
MAT READ c()
MAT PRINT b(),5,1
PRINT STRING$(18,"-")
MAT PRINT c(),5,1
```

```
PRINT STRING$(18,"-")
MAT MUL a()=b()*c()
MAT PRINT a(),5,1

--> Gibt     1.0,  2.0, -3.0
           ------------------
             4.0,  5.0, -1.0
           ------------------
             4.0,  5.0, -1.0
             8.0, 10.0, -2.0
           -12.0,-15.0,  3.0     aus.
```

MAT MUL x=a()*b() ist nur für Vektoren mit gleicher Elementenzahl definiert. Als Ergebnis x erhält man das Skalarprodukt (sog. inneres Produkt) der Vektoren a und b. Das Skalarprodukt zweier Vektoren ist definiert als die Summe der n Produkte a(i)*b(i), i=1,...,n.

Kürzel: m x=a()*b()

Beispiel:

```
DIM b(3),c(3)
DATA 1,2,-3,4,5,-1
MAT READ b()
MAT READ c()
MAT PRINT b(),5,1
PRINT STRING$(18,"-")
MAT PRINT c(),5,1
PRINT STRING$(18,"-")
MAT MUL x=b()*c()
PRINT x

--> Gibt     1.0,  2.0, -3.0
           ------------------
             4.0,  5.0, -1.0
           ------------------
           17.0                    aus.
```

MAT MUL x=a()*b()*c() ist definiert für 'geeignete' Vektoren a und c sowie eine 'geeignete' Matrix b(). Das Ergebnis ist der Skalar x, der sich ergibt, wenn man den (Zeilen-)Vektor a mit der Matrix b und dieses Ergebnis dann mit dem (Spalten-)Vektor c multipliziert. Der Vektor a muß dazu genausoviele Elemente besitzen wie die Matrix b Zeilen aufweist. Der Vektor c muß genausoviele Elemente besitzen wie die Matrix b Spalten aufweist, z.B.

```
DIM a(5),b(5,3),c(3).
```

Kürzel: m x=a()*b()*c()

Beispiel:

```
DIM a(2),b(2,3),c(3)
DATA 1,2,-3,4,5
MAT READ a()
MAT READ c()
MAT SET b()=1
MAT PRINT a(),5,1
PRINT STRING$(18,"-")
MAT PRINT b(),5,1
PRINT STRING$(18,"-")
MAT PRINT c(),5,1
PRINT STRING$(18,"-")
MAT MUL x=a()*b()*c()
PRINT x
```

```
--> Gibt      1.0,  2.0
             ------------------
              1.0,  1.0,  1.0
              1.0,  1.0,  1.0
              1.0,  1.0,  1.0
             ------------------
             -3.0,  4.0,  5.0
             ------------------
             18.0                    aus.
```

MAT NORM a(),0 bzw. MAT NORM a(),1 sind definiert für Matrizen und Vektoren. MAT NORM a(),0 normiert eine Matrix (einen Vektor) zeilen-, MAT NORM a(),1 normiert eine Matrix (einen Vektor) spaltenweise. Dieses bedeutet, daß im Falle der zeilenweisen (spaltenweisen) Normierung die Summe der Quadrate aller Elemente in jeder Zeile (Spalte) identisch gleich 1 ist.

Kürzel: m no a(),0 bzw. m no a(),1

Beispiel:

```
DIM a(10,10),b(10,10),v(10)
DATA 1,2,3,4,5,6,7,8,9,-1
DATA 3.2,4,-5,2.4,5.1,6.2,7.2,8.1,6,-5
DATA -2,-5,-6,-1.2,-1.5,-6.7,4.5,8.1,3.4,10
DATA 5,-2.3,4,5.6,12.2,18.2,14.1,16,-21,-13
DATA 4.1,5.2,16.7,18.4,19.1,20.2,13.6,14.8,19.4,18.6
DATA 15.2,-1.8,13.6,-4.9,5.4,19.8,16.4,-20.9,21.4,13.8
DATA -3.6,6,-8.2,-9.1,4,-2.5,2,3.4,6.7,8.4
DATA 4.7,8.3,9.4,10.5,11,19,15.4,18.9,-20,12.6
DATA 5.3,-4.7,6.1,6.5,6.9,-9.2,-10.8,4.3,5.6,9.1
DATA 21.4,19.5,28.4,19.3,24.6,14.9,71.3,23.5,14.5,-12.3
'
CLS
MAT READ a()
MAT CPY b()=a()          ! Ursprungsmatrix sichern
PRINT "Ursprungsmatrix"
```

```
PRINT
MAT PRINT a(),7,2
~INP(2)
'
' zeilenweise Normierung
'
CLS
MAT NORM a(),0
PRINT
PRINT "zeilenweise normiert : "
PRINT
MAT PRINT a(),7,2
~INP(2)
'
' Überprüfung der zeilenweisen Normierung
'
PRINT
PRINT "Probe : "
PRINT
FOR i%=1 TO 10
  MAT XCPY v()=a(i%,1)  ! kopiert a() zeilenweise in den Vektor v()
  MAT MUL x=v()*v()     ! berechnet das Skalarprodukt zwischen v() und
                          v()
  PRINT x'
NEXT i%
PRINT
~INP(2)
'
' spaltenweise Normierung
'
CLS
MAT CPY a()=b()         ! Ursprungsmatrix zurückkopieren
MAT NORM a(),1
PRINT "spaltenweise normiert : "
PRINT
MAT PRINT a(),7,2
~INP(2)
```

```
'
' Überprüfung der spaltenweisen Normierung
'
PRINT
PRINT "Probe : "
PRINT
FOR i%=1 TO 10
  MAT CPY v()=a(1,i%)  ! kopiert a() spaltenweise in den Vektor v()
  MAT MUL x=v()*v()    ! berechnet das Skalarprodukt zwischen v() und
                         v()
  PRINT x'
NEXT i%
~INP(2)
```

--> Gibt

Ursprungsmatrix

```
  1.00,   2.00,   3.00,   4.00,   5.00,   6.00,   7.00,   8.00,   9.00,  -1.00
  3.20,   4.00,  -5.00,   2.40,   5.10,   6.20,   7.20,   8.10,   6.00,  -5.00
 -2.00,  -5.00,  -6.00,  -1.20,  -1.50,  -6.70,   4.50,   8.10,   3.40,  10.00
  5.00,  -2.30,   4.00,   5.60,  12.20,  18.20,  14.10,  16.00, -21.00, -13.00
  4.10,   5.20,  16.70,  18.40,  19.10,  20.20,  13.60,  14.80,  19.40,  18.60
 15.20,  -1.80,  13.60,  -4.90,   5.40,  19.80,  16.40, -20.90,  21.40,  13.80
 -3.60,   6.00,  -8.20,  -9.10,   4.00,  -2.50,   2.00,   3.40,   6.70,   8.40
  4.70,   8.30,   9.40,  10.50,  11.00,  19.00,  15.40,  18.90, -20.00,  12.60
  5.30,  -4.70,   6.10,   6.50,   6.90,  -9.20, -10.80,   4.30,   5.60,   9.10
 21.40,  19.50,  28.40,  19.30,  24.60,  14.90,  71.30,  23.50,  14.50, -12.30
```

zeilenweise normiert:

```
 0.06,  0.12,  0.18,  0.24,  0.30,  0.35,  0.41,  0.47,  0.53, -0.06
 0.18,  0.23, -0.29,  0.14,  0.29,  0.36,  0.42,  0.47,  0.35, -0.29
-0.11, -0.28, -0.34, -0.07, -0.09, -0.38,  0.26,  0.46,  0.19,  0.57
 0.12, -0.06,  0.10,  0.14,  0.30,  0.45,  0.35,  0.40, -0.52, -0.32
 0.08,  0.10,  0.33,  0.36,  0.38,  0.40,  0.27,  0.29,  0.38,  0.37
 0.32, -0.04,  0.29, -0.10,  0.11,  0.42,  0.35, -0.44,  0.45,  0.29
-0.19,  0.32, -0.44, -0.48,  0.21, -0.13,  0.11,  0.18,  0.36,  0.45
 0.11,  0.19,  0.21,  0.24,  0.25,  0.43,  0.35,  0.43, -0.46,  0.29
 0.23, -0.21,  0.27,  0.29,  0.31, -0.41, -0.48,  0.19,  0.25,  0.40
 0.23,  0.21,  0.30,  0.21,  0.26,  0.16,  0.76,  0.25,  0.15, -0.13
```

Probe:

```
1 1 1 1 1 1 1 1 1 1
```

spaltenweise normiert:

```
 0.04,  0.08,  0.08,  0.12,  0.13,  0.14,  0.09,  0.18,  0.20, -0.03
 0.11,  0.16, -0.13,  0.07,  0.14,  0.14,  0.09,  0.18,  0.13, -0.14
-0.07, -0.21, -0.15, -0.04, -0.04, -0.15,  0.06,  0.18,  0.07,  0.28
 0.18, -0.09,  0.10,  0.17,  0.33,  0.41,  0.18,  0.35, -0.46, -0.36
 0.14,  0.21,  0.42,  0.57,  0.51,  0.46,  0.17,  0.33,  0.42,  0.52
 0.53, -0.07,  0.35, -0.15,  0.15,  0.45,  0.21, -0.46,  0.47,  0.38
-0.13,  0.25, -0.21, -0.28,  0.11, -0.06,  0.03,  0.08,  0.15,  0.23
 0.17,  0.34,  0.24,  0.33,  0.30,  0.43,  0.20,  0.42, -0.44,  0.35
 0.19, -0.19,  0.15,  0.20,  0.19, -0.21, -0.14,  0.10,  0.12,  0.25
 0.75,  0.80,  0.72,  0.60,  0.66,  0.34,  0.90,  0.52,  0.32, -0.34
```

Probe:

```
1 1 1 1 1 1 1 1 1 1          aus.
```

MAT DET x=a([i,j])[,n] berechnet die Determinante einer quadratischen Matrix vom Typ (n,n). Dabei ist der Zeilen- und Spaltenoffset abhängig von MAT BASE 0 bzw. MAT BASE 1 auf a(0,0) bzw. a(1,1) voreingestellt,

sofern OPTION BASE 1 gilt. Es kann jedoch auch die Determinante einer quadratischen Teilmatrix berechnet werden. Dazu muß durch i und j der Zeilen- und Spaltenoffset von a() und durch n die Anzahl der Elemente der Teilmatrix angegeben werden. Dabei wird intern eine ab der 'Position' i-te Zeile, j-te Spalte eine Teilmatrix vom Typ (n,n) erzeugt.

Kürzel: m d x=a([i,j])[,n]

Beispiel:

```
DIM a(10,10),b(4,4)
DATA 1,2,3,4,5,6,7,8,9,-1
DATA 3.2,4,-5,2.4,5.1,6.2,7.2,8.1,6,-5
DATA -2,-5,-6,-1.2,-1.5,-6.7,4.5,8.1,3.4,10
DATA 5,-2.3,4,5.6,12.2,18.2,14.1,16,-21,-13,3.8
DATA 4.1,5.2,16.7,18.4,19.1,20.2,13.6,14.8,19.4,18.6
DATA 15.2,-1.8,13.6,-4.9,5.4,19.8,16.4,-20.9,21.4,13.8
DATA -3.6,6,-8.2,-9.1,4,-2.5,2,3.4,6.7,8.4,10.9
DATA 4.7,8.3,9.4,10.5,11,19,15.4,18.9,-20,12.6
DATA 5.3,-4.7,6.1,6.5,6.9,-9.2,-10.8,4.3,5.6,9.1
DATA 21.4,19.5,28.4,19.3,24.6,14.9,71.3,23.5,14.5,-12.3
'
CLS
MAT READ a()
PRINT "Ursprungsmatrix"
PRINT
MAT PRINT a(),7,2
PRINT
PRINT "Determinante : ";
MAT DET x=a()           ! Berechnet die Determinante von a()
PRINT x;
MAT DET y=a(1,4),4      ! Berechnet die Determinante der Teilmatrix
PRINT
PRINT "Determinante von a(1,4),4 : ";
PRINT y
PRINT
PRINT "Probe :"
```

```
PRINT
MAT CPY b()=a(1,4),4,4 ! kopiert die Teilmatix nach b()
MAT PRINT b(),7,2
MAT DET z=b()          ! berechnet die Determinante von b()
PRINT
PRINT z
```

--> Gibt

Ursprungsmatrix

```
  1.00,   2.00,   3.00,   4.00,   5.00,   6.00,   7.00,   8.00,   9.00,  -1.00
  3.20,   4.00,  -5.00,   2.40,   5.10,   6.20,   7.20,   8.10,   6.00,  -5.00
 -2.00,  -5.00,  -6.00,  -1.20,  -1.50,  -6.70,   4.50,   8.10,   3.40,  10.00
  5.00,  -2.30,   4.00,   5.60,  12.20,  18.20,  14.10,  16.00, -21.00, -13.00
  3.80,   4.10,   5.20,  16.70,  18.40,  19.10,  20.20,  13.60,  14.80,  19.40
 18.60,  15.20,  -1.80,  13.60,  -4.90,   5.40,  19.80,  16.40, -20.90,  21.40
 13.80,  -3.60,   6.00,  -8.20,  -9.10,   4.00,  -2.50,   2.00,   3.40,   6.70
  8.40,  10.90,   4.70,   8.30,   9.40,  10.50,  11.00,  19.00,  15.40,  18.90
-20.00,  12.60,   5.30,  -4.70,   6.10,   6.50,   6.90,  -9.20, -10.80,   4.30
  5.60,   9.10,  21.40,  19.50,  28.40,  19.30,  24.60,  14.90,  71.30,  23.50

Determinante:                -549840202186
Determinante von a(1,4),4: -57.61200000001
```

Probe:

```
  4.00,   5.00,   6.00,   7.00
  2.40,   5.10,   6.20,   7.20
 -1.20,  -1.50,  -6.70,   4.50
  5.60,  12.20,  18.20,  14.10

-57.61200000001          aus.
```

MAT QDET x=a([i,j])[,n] funktioniert wie MAT DET x = a([i,j])[,n], ist jedoch geschwindigkeits- und nicht genauigkeitsoptimiert. In der Regel

liefern beide Verfahren das gleiche Ergebnis. Bei 'kritischen' Matrizen, deren Determinante nahe bei 0 liegt, sollte jedoch stets MAT DET verwendet werden.

Kürzel: M qd x=a([i,j])[,n]

Beispiel:

```
DIM a(10,10)
DATA 1,2,3,4,5,6,7,8,9,-1
DATA 3.2,4,-5,2.4,5.1,6.2,7.2,8.1,6,-5
DATA -2,-5,-6,-1.2,-1.5,-6.7,4.5,8.1,3.4,10
DATA 5,-2.3,4,5.6,12.2,18.2,14.1,16,-21,-13,3.8
DATA 4.1,5.2,16.7,18.4,19.1,20.2,13.6,14.8,19.4,18.6
DATA 15.2,-1.8,13.6,-4.9,5.4,19.8,16.4,-20.9,21.4,13.8
DATA -3.6,6,-8.2,-9.1,4,-2.5,2,3.4,6.7,8.4,10.9
DATA 4.7,8.3,9.4,10.5,11,19,15.4,18.9,-20,12.6
DATA 5.3,-4.7,6.1,6.5,6.9,-9.2,-10.8,4.3,5.6,9.1
DATA 21.4,19.5,28.4,19.3,24.6,14.9,71.3,23.5,14.5,-12.3
'
CLS
MAT READ a()
PRINT "Ursprungsmatrix"
PRINT
MAT PRINT a(),7,2
PRINT
PRINT "Determinante mit  MAT DET : ";
MAT DET x=a()
PRINT x;
PRINT
PRINT "Determinante mit MAT QDET : ";
MAT DET y=a()
PRINT y;
PRINT
PRINT "Abweichung : ";x-y
```

```
--> Gibt
```

Ursprungsmatrix

```
  1.00,   2.00,   3.00,   4.00,   5.00,   6.00,   7.00,   8.00,   9.00,  -1.00
  3.20,   4.00,  -5.00,   2.40,   5.10,   6.20,   7.20,   8.10,   6.00,  -5.00
 -2.00,  -5.00,  -6.00,  -1.20,  -1.50,  -6.70,   4.50,   8.10,   3.40,  10.00
  5.00,  -2.30,   4.00,   5.60,  12.20,  18.20,  14.10,  16.00, -21.00, -13.00
  3.80,   4.10,   5.20,  16.70,  18.40,  19.10,  20.20,  13.60,  14.80,  19.40
 18.60,  15.20,  -1.80,  13.60,  -4.90,   5.40,  19.80,  16.40, -20.90,  21.40
 13.80,  -3.60,   6.00,  -8.20,  -9.10,   4.00,  -2.50,   2.00,   3.40,   6.70
  8.40,  10.90,   4.70,   8.30,   9.40,  10.50,  11.00,  19.00,  15.40,  18.90
-20.00,  12.60,   5.30,  -4.70,   6.10,   6.50,   6.90,  -9.20, -10.80,   4.30
  5.60,   9.10,  21.40,  19.50,  28.40,  19.30,  24.60,  14.90,  71.30,  23.50

Determinante mit  MAT DET: -2549840202186
Determinante mit MAT QDET: -2549840202186
Abweichung : 0                                    aus.
```

MAT RANG x=a([i,j])[,n] gibt den Rang einer quadratischen Matrix aus. Dabei kann wie bei MAT DET bzw. bei MAT QDET ein beliebiger Zeilen- und Spaltenoffset gewählt werden. Die Anzahl der Elemente der Teilmatrix ist durch n anzugeben. Dabei wird intern ab der 'Position' i-te Zeile, j-te Spalte eine Teilmatrix vom Typ (n,n) erzeugt.

Kürzel: m ra x=a([i,j])[,n]

Beispiel:

```
DIM a(5,5)
DATA 1,2,3,4,5
DATA 3.2,4,-5,2.4,5.1
DATA -2,4,-5,2.4,5.1
DATA 5,-2.3,4,5.6,12.2
DATA 4.1,5.2,16.7,18.4,19.1
'
```

```
CLS
MAT READ a()
PRINT "Ursprungsmatrix"
PRINT
MAT PRINT a(),7,2
PRINT
PRINT "Rang von a(): ";
MAT RANG x=a()
PRINT x;
PRINT
PRINT "Rang von a(1,2),3 : ";
MAT RANG y=a(1,2),3
PRINT y;
PRINT
```

--> Gibt

Ursprungsmatrix

```
   1.00,   2.00,   3.00,   4.00,   5.00
   3.20,   4.00,  -5.00,   2.40,   5.10
  -2.00,   4.00,  -5.00,   2.40,   5.10
   5.00,  -2.30,   4.00,   5.60,  12.20
   4.10,   5.20,  16.70,  18.40,  19.10

Rang von a():       5
Rang von a(1,2),3: 2     aus.
```

MAT INV b() = a() dient zur Bestimmung der Inversen einer quadratischen Matrix. Die Inverse der Matrix a() wird in die Matrix b() geschrieben. b() muß daher vom gleichen Typ wie a() sein.

Kürzel : m inv b() = a()

Beispiel:

```
DIM a(5,5),b(5,5),c(5,5)
DATA 1,2,3,4,5
DATA 3.2,4,-5,2.4,5.1
DATA -2,4,-5,2.4,5.1
DATA 5,-2.3,4,5.6,12.2
DATA 4.1,5.2,16.7,18.4,19.1
'
CLS
MAT READ a()
PRINT "Ursprungsmatrix a() : "
PRINT
MAT PRINT a(),7,2
'
MAT INV b()=a()
PRINT
PRINT "Inverse von a() : "
PRINT
MAT PRINT b(),7,2
PRINT
PRINT "Probe b()*a() = Einheitsmatrix ? "
PRINT
MAT MUL c()=b()*a()
MAT PRINT c(),7,2
```

--> Gibt

Ursprungsmatrix a()

```
  1.00,   2.00,   3.00,   4.00,   5.00
  3.20,   4.00,  -5.00,   2.40,   5.10
 -2.00,   4.00,  -5.00,   2.40,   5.10
  5.00,  -2.30,   4.00,   5.60,  12.20
  4.10,   5.20,  16.70,  18.40,  19.10
```

Inverse von a():

```
 0.00,   0.19,  -0.19,  -0.00,  -0.00
 0.97,   0.02,  -0.09,  -0.10,  -0.17
 0.71,  -0.10,  -0.10,  -0.01,  -0.12
-1.65,   0.17,   0.11,  -0.06,   0.39
 0.71,  -0.12,   0.04,   0.09,  -0.17
```

Probe b()*a() = Einheitsmatrix?

```
 1.00,   0.00,   0.00,   0.00,   0.00
 0.00,   1.00,   0.00,   0.00,  -0.00
 0.00,  -0.00,   1.00,   0.00,  -0.00
-0.00,  -0.00,  -0.00,   1.00,   0.00
-0.00,   0.00,   0.00,   0.00,   1.00          aus.
```

14 - Weitere Befehle der Version 3.5

In der Version 3.5 des GFA-BASIC wurden - neben den in Kapitel 13 beschriebenen Befehlen - auch drei Befehle aus dem Bereich der Kombinatorik, zwei Befehle zur Operation mit DATA-Zeigern, den Befehl PCOLOR für die Bildschirmausgabe, die Befehle GURU x, GURU, eine neue Variante des Befehls OPENW sowie _0 bis _9 implementiert.

Befehle aus dem Bereich der Kombinatorik

Die Befehle aus dem Bereich der Kombinatorik lauten:

x=FACT(n),
y=VARIAT(n,k)
z=COMBIN(n,k)

```
x,y,z:  aexp
  n,k:  iexp
```

x=FACT(n) berechnet die Fakultät (n!) von n und schreibt diesen Wert in die Variable x. Die Fakultät einer natürlichen Zahl n ist definiert als als das Produkt über die ersten n natürlichen Zahlen, mit 0! = 1.

y=VARIAT(n,k) berechnet die Anzahl der Variationen von n Elementen zur k-ten Klasse ohne Wiederholung und schreibt diesen Wert in die Variable y.

Die Anzahl der Variationen von n Elementen zur k-ten Klasse ohne Wiederholung ist definiert als

```
VARIAT(n,k) = n!/(n-k)!
```

z=COMBIN(n,k) berechnet die Anzahl der Kombinationen von n Elementen zur k-ten Klasse ohne Wiederholung und schreibt diesen Wert in die Variable z. Die Anzahl der Kombinationen von n Elementen zur k-ten Klasse ohne Wiederholung ist definiert als

```
COMBIN(n,k) = n!/((n-k)!*k!)
```

Beispiel:

```
x=FACT(6)
y=VARIAT(6,2)
z=COMBIN(6,2)
PRINT x,y,z
~INP(2)

--> Gibt     720     30     15     aus.
```

Neue Variante des Befehls OPENW

Wenn die Screen-Nummer negativ ist, dann wird das Fenster im aktuellen Screen geöffnet (= Screen oben).

Beispiel:

```
OPENS 4
DO
  REPEAT
  UNTIL MOUSEK=3
  OPENW #1,100,100,80,80,0,15,-1
  CLOSEW #1
LOOP
```

Bestimmen von Vorder- und Hintergrundfarbe

PCOLOR foreground, background

Dieser Befehl führt das gleiche aus wie COLOR, für die Bildschirmausgabe wird jedoch der PRINT-Befehl verwendet.

Beispiel:

```
PCOLOR   2,3
PRINT    TIME$
PCOLOR   1
PRINT    DATE$
PCOLOR   0,2
PRINT    "HALLO"
```

Befehle zur Operation mit DATA-Zeigern

_DATA
_DATA=

_DATA gibt an, wo der DATA-Zeiger steht. _DATA ist 0, falls das nächste READ ein "out of data" ergeben würde.

_DATA= erlaubt das Setzen des DATA-Zeigers auf einen vorher mit _DATA ermittelten Wert.

Beispiel:

```
DIM dp%(100)
DATA 1,2,3,4,5,6,7,8,9
DATA 13,24,328,3242,1,0
'
i%=0
DO WHILE _DATA
  dp%(i%)=_DATA
  INC i%
  READ a
LOOP
'
DEC i%
'
FOR j%=i% DOWNTO 0
  _DATA= dp%(j%)
  READ a
  PRINT a'
NEXT j%
~INP(2)

--> Gibt  0 1 3242 328 24 13 9 8 7 6 5 4 3 2 1  aus.
```

Spezielle Variablen

_0..._9

Die Befehle _0 bis _9 sind zehn spezielle Variablen. Es handelt sich hierbei um Langwörter wie %, also nicht um Fließkommazahlen.

Diese Variablen werden nur mit dem Befehl 'NEW' gelöscht, und nicht mit CLEAR, LOAD, CHAIN oder RUN.

Man kann diese Variablen dazu verwenden, um z.B. in der Testphase eines Programms sich AllocMem-Adressen, Library-Base oder Ähnliches zu merken und auch bei einem RUN korrekt freizugeben bzw. über ein CHAIN zu retten.

GURU x

wenn x = 0: keine GURU-Abfrage

wenn x < > 0: Beim Erscheinen einer GURU Meditation (= Systemabsturz) springt GFA-BASIC in die ONERROR GOSUB-Routine ein oder es wird die Fehlermeldung #109 "GURU Meditation!" erzeugt.

Die wichtigsten GURU-Nummern können mit der Funktion GURU abgefragt werden.

Beispiel:

```
GURU 1                    ! Schaltet GURU-Abfrage ein
ONERROR GOSUB ER
~FREEMem(1000000,10)      ! Erzeugt GURU
PROCEDURE ER
  PRINT HEX$(GURU)
  END
RETURN
```

Anmerkungen:

- Die Fehlermeldungen #110 bis #118 sind die 'normalen' GURUS $2 bis $A.

- Im compilierten Programm (Version 3.5) ist es **nicht** möglich, ein "RESUME NEXT" oder "RESUME LABEL" durchzuführen, wenn ein GURU einen ONERROR GOSUB-Aufruf verursacht hat.
  Am sichersten ist es immer, nach der Fehlermeldung #109 ("GURU Meditation") das Programm so schnell wie möglich zu beenden. Die GURUS $2 bis $A (Fehlermeldung #110 bis #118) sind nicht so gefährlich und stellen normalerweise keine Probleme dar.

- Die GURU-Abfrage funktioniert nur, wenn es für das System überhaupt noch möglich ist, weiter zu laufen. GURUS sind eigentlich System-Probleme und keine GFA-BASIC-Probleme!

GFA Systemtechnik GmbH
Heerdter Sandberg 30-32
D-4000 Düsseldorf 11
Telefon 02 11/55 04-0